

Apple.

AppleTalk. for VMS

Protocol Support Library
Reference Manual

 APPLE COMPUTER, INC.

This manual and the software described in it are copyrighted, with all rights reserved. Under the copyright laws, this manual may not be copied, in whole or in part, without written consent of Apple Computer. Under the law, copying includes translating into another language or format.

© Apple Computer, Inc., 1987
20525 Mariani Avenue
Cupertino, California 95014
(408) 996-1010

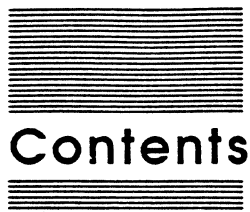
Apple, the Apple logo, LaserWriter, and AppleTalk are registered trademarks of Apple Computer, Inc.

Macintosh is a trademark of Apple Computer, Inc.

Digital, DECnet, VAX, VMS, BLISS, and DIBOL are trademarks of Digital Equipment Corporation.

Ethernet is a trademark of Xerox Corporation.

KFPS, KFPQ, and FastPath are trademarks of Kinetics Inc.



Contents

Figures and tables vi

Preface About This Manual vii

Intended audience vii

Structure of this document viii

Associated documents viii

Conventions used in this document ix

Part I Protocol Library Overview 1

Chapter 1 Using the Protocol Library 3

Features of the Protocol Library 4

Organization of the Protocol Library 5

How to call Protocol Library procedures 6

Call formats 7

Naming conventions 7

Argument passing 8

Passing arguments by value 8

Passing arguments by reference 9

Passing arguments by descriptor 9

Event flag usage 9

Dynamic memory allocation 10

Errors from Protocol Library procedures 10

Linking with the Protocol Library 10

Chapter 2 Low-Level I/O Facilities 11

Link-level communication and the buffer pool 12

Incoming datagrams 13

Sending datagrams 14

Tuning buffer allocation 14

Devices and ports 14

Port drivers 15

Device types 15

Communicating on ports 16

Virtual network (VLAP) 16

Ethernet (ELAP) 17

Asynchronous half-bridge (AHLAP) 17

DECnet (DLAP) 17

Part II Protocol Library Routines 19

ATK\$ATP_CANCEL_REQ - Cancel ATP request 21
ATK\$ATP_CANCEL_RSP - Cancel ATP response 22
ATK\$ATP_CLS_SKT - Close an ATP socket 23
ATK\$ATP_GET_REQ - Receive an ATP request 24
ATK\$ATP_OPN_SKT - Open an ATP socket 28
ATK\$ATP_SND_REQ - Send an ATP request 29
ATK\$ATP_SND_RSP - Send an ATP response 34
ATK\$BYTESWAP - Byte swapper 39
ATK\$CHECKSUM - Compute DDP checksum 41
ATK\$CLOSE_PORT - Close a port driver 42
ATK\$DDP_CLS_SKT - Close a DDP socket 43
ATK\$DDP_OPN_SKT - Open a DDP socket 44
ATK\$DDP_READ - Read a DDP datagram 46
ATK\$DDP_WRITE - Transmit a DDP datagram 47
ATK\$ENTITY_LEN - Return length of NBP entity 51
ATK\$EP_REQ - Send an EP request 52
ATK\$GET_DATAGRAM - Get a datagram buffer 55
ATK\$GET_PHYS_ADDR - Get port physical address 56
ATK\$INIT_ATALK - Initialize AppleTalk 57
ATK\$INIT_PORT - Load and initialize a port driver 59
ATK\$INSQUE - Insert entry in queue 60
ATK\$LAP_XMIT - Transmit a datagram 61
ATK\$MAKE_ENTITY - Construct an NBP entity 64
ATK\$NBP_CONFIRM - Confirm name via NBP 66
ATK\$NBP_LOOKUP - Look up name(s) via NBP 69
ATK\$NBP_REGISTER - Register name via NBP 73
ATK\$NBP_REMOVE - Remove NBP name 75
ATK\$NET_NODE - Get AppleTalk address 77
ATK\$OPEN_PORT - Open port 78
ATK\$PAP_CLOSE - Close a PAP connection 80
ATK\$PAP_OPEN - Open a PAP connection 81
ATK\$PAP_READ - Read over a PAP connection 87
ATK\$PAP_REG_NAME - Register a server name 90
ATK\$PAP_REM_NAME - Remove a server name 92
ATK\$PAP_SLINIT - Initialize a PAP server 93
ATK\$PAP_STATUS - Request PAP server status 99
ATK\$PAP_WRITE - Write over PAP connection 103
ATK\$REMQUE - Remove entry from queue 106
ATK\$RETURN_DATAGRAM - Return datagram to silo 107
ATK\$RTMP_REQ - Broadcast an RTMP request 108
ATK\$TMRINS - Install a timer process 110
ATK\$TMRREM - Remove a timer process 112
ATK\$TMRRESET - Reset a timer process 113
ATK\$VA_COUNT - Variable argument count 114
ATK\$ZIP_BRINGUP - Bring up a network 115
ATK\$ZIP_GET_MY_ZONE - Get port's zone name 116
ATK\$ZIP_GET_ZONE_LIST - Get list of zone names in
internet 117
ATK\$ZIP_QUERY - Send network zone name query 119
ATK\$ZIP_TAKEDOWN - Take down a network 121

Appendix A Port Drivers 123

ATKAHLAP – Asynchronous half-bridge port driver 124

ATKELAP – Ethernet port driver 126

ATKVLAP – Virtual AppleTalk port driver 128

ATKDLAP – DECnet half-bridge port driver 130

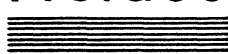
Index 135

Figures and tables

Figure 1-1	Organization of the Protocol Library	6
Figure 2-1	Using a circular buffer list as a silo (FIFO)	13
Table 2-1	Supported communication devices	16



Preface



About This Manual

This manual provides developers of AppleTalk® applications for the VAX/VMS operating system with detailed usage and reference information about AppleTalk for VMS Protocol Support Library routines.

Protocol Library routines can only be used in programs written in languages that produce native code for the VAX hardware, and operate in conjunction with the VMS operating system. At present, these languages include VAX MACRO and the following high-level languages:

- ☐ VAX BASIC ☐ VAX BLISS-32
- ☐ VAX C ☐ VAX COBOL
- ☐ VAX COBOL-74 ☐ VAX CORAL
- ☐ VAX DIBOL ☐ VAX FORTRAN
- ☐ VAX PASCAL ☐ VAX PL/I
- ☐ VAX RPG

Language systems not supplied by Digital Equipment Corporation may be able to use the Protocol Library routines if they adhere to the VAX Procedure Calling and Condition Handling Standard.

Intended audience

This manual is intended for system and application programmers who want to use the Protocol Library routines. It assumes that the reader has a working knowledge of the AppleTalk network architecture and VAX/VMS system or application programming experience. This is *not* a tutorial.

Structure of this document

This manual is organized into two parts, as follows:

- Part I provides background information, instructions, and guidelines on the use of the Protocol Library for AppleTalk communication on VMS.
 - Chapter 1 describes the organization of the Protocol Library, presents an overview of its features, and describes how to call and link with a library procedure.
 - Chapter 2 describes the low-level I/O facilities of the Protocol Library.
- Part II provides detailed reference information on each of the AppleTalk Protocol Library routines, in alphabetical order and closely following the documentation format contained in the *VAX/VMS Run-Time Library Routines Reference Manual* from Digital Equipment Corporation.

Associated documents

The following manuals from Digital Equipment Corporation document the VAX/VMS system services, run-time libraries, and I/O architecture, and describe the documentation standards used throughout the VMS manuals.

- *Introduction to VAX/VMS System Routines*
- *VAX/VMS Run-Time Library Routines Reference Manual*
- *VAX/VMS System Services Reference Manual*
- *Guide to Writing a Device Driver for VAX/VMS*
- *I/O User's Guide*
- *Programming in VAX C*
- ❖ *Note:* The Documentation Standard described in the *VAX/VMS Run-Time Library Routines Reference Manual* is recommended reading prior to attempting to use this manual as a reference.

The following manuals from Apple Computer, Inc. document the AppleTalk network architecture, the interface provided to client processes of AppleTalk, and the architecture of the AppleTalk for VMS system.

- Sidhu, Gursharan S., Richard F. Andrews, and Alan B. Oppenheimer, *Inside AppleTalk* (Apple Programmer's and Developer's Association)
- *Inside Macintosh*, Volume II, Chapter 10, "The AppleTalk Manager"
- *Inside Macintosh*, Volume IV, Chapter 23, "The AppleTalk Manager"
- *Inside Macintosh*, Volume V, Chapter 28, "The AppleTalk Manager"
- *AppleTalk for VMS Architecture and Implementation*
- *AppleTalk for VMS Installation and Operation Guide*
- *AppleTalk for VMS Bridge Control Program Guide*

Conventions used in this document

Convention	Meaning
file-spec,...	Horizontal ellipsis indicates that <i>additional</i> parameters, values, or information can be entered or supplied
[logical-name]	Square brackets indicate that the enclosed item is <i>optional</i> (except when these brackets are used in a file name; for example, DUA0: [SYS0.SYSMGR] PORTDESC.TEXT)
quotation mark	Used to refer to the "double quote" (")
apostrophe	Used to refer to the "single quote" (')

Other conventions used in this document are described in the VAX/VMS manuals referred to in the previous section, particularly the *VAX/VMS Run-Time Library Routines Reference Manual*.

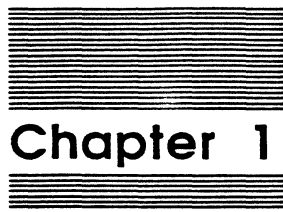




Part I

Protocol Library Overview





Chapter 1

Using the Protocol Library

The AppleTalk for VMS Protocol Support Library (or simply the Protocol Library) provides programmers with a coordinated set of procedures for accessing AppleTalk services under VAX/VMS. The library provides the functionality of the Macintosh AppleTalk Manager, but the details of the interface differ due to the widely different architectures of the Macintosh and VAX/VMS.

The Protocol Library provides a common run-time environment for applications because its procedures follow the VAX Modular Programming Standard. The advantage of a common run-time environment is that any program written in MACRO, BLISS, or another supported high-level language can call any procedure in the Protocol Library.

A **procedure** is a set of related instructions that performs a particular task. It is an executable program unit and can be a main program, subroutine, or function. A procedure has an entry point and an argument list. It may also return a function value or completion status to the program that calls it.

❖ *Note:* In the Protocol Library documentation, procedures are also referred to as Protocol Library **routines**.

Protocol Library procedures are written using the VAX Procedure Calling and Condition Handling Standard as described in the *Introduction to VAX/VMS System Routines*. They are AST reentrant and position-independent. They may generally be used in conjunction with VAX/VMS system services, which are also callable procedures (see the *VAX/VMS System Services Reference Manual*).

Features of the Protocol Library

The Protocol Library provides the following features and capabilities:

- ❑ Protocol Library procedures provide a rich set of AppleTalk operations at the same level as the AppleTalk Manager for the Macintosh.
- ❑ Access is provided to currently documented levels of the AppleTalk protocol architecture, including DDP, ATP, NBP, ZIP, EP, and PAP. The modularity of the library permits addition of further protocol support at a later time.
- ❑ Incoming datagrams are placed in buffers that are allocated from a process-wide buffer pool. This eliminates buffer copying and provides a datagram silo facility, which can smooth the flow of incoming datagrams as they are routed to destination sockets.
- ❑ Each procedure is language-independent, so that a separate procedure need not be written for each language implementation. Similarly, the results of a Protocol Library call will be the same irrespective of what language is used to call it.
- ❑ The library is implemented as a shareable image, thereby reducing memory requirements when multiple processes are using the library procedures.
- ❑ No privileged mode (supervisor, executive, or kernel mode) access is required, protecting the system from damage by a malfunctioning library or program.
- ❑ The library is vectored, making it unnecessary to relink applications that use the shareable image after a change or addition is made to the library. This makes it possible to add AppleTalk protocols to the library as they become stabilized, without relinking programs used with earlier versions of the library.
- ❑ Library procedures can be called from both the AST and non-AST levels.

Organization of the Protocol Library

The Protocol Library contains procedures that implement the AppleTalk protocols specified below and shown in Figure 1-1. The Datagram Delivery Protocol (DDP) and AppleTalk Transaction Protocol (ATP) correspond to the Network and Transport layers of the International Standards Organization (ISO) Open Systems Interconnection (OSI) model. The socket-level routing, sending, and receive demultiplexing are performed within the Protocol Library by the DDP procedures.

The library routines are organized into the following groups:

- ☐ low-level I/O, initialization, and cleanup
- ☐ buffer pool management
- ☐ Datagram Delivery Protocol (DDP) services (including a choice of full or stub routing)
- ☐ AppleTalk Transaction Protocol (ATP) services
- ☐ Name Binding Protocol (NBP) services
- ☐ Routing Table Maintenance Protocol (RTMP) stub
- ☐ Zone Information Protocol (ZIP) services
- ☐ Echo Protocol (EP) services
- ☐ Printer Access Protocol (PAP) services

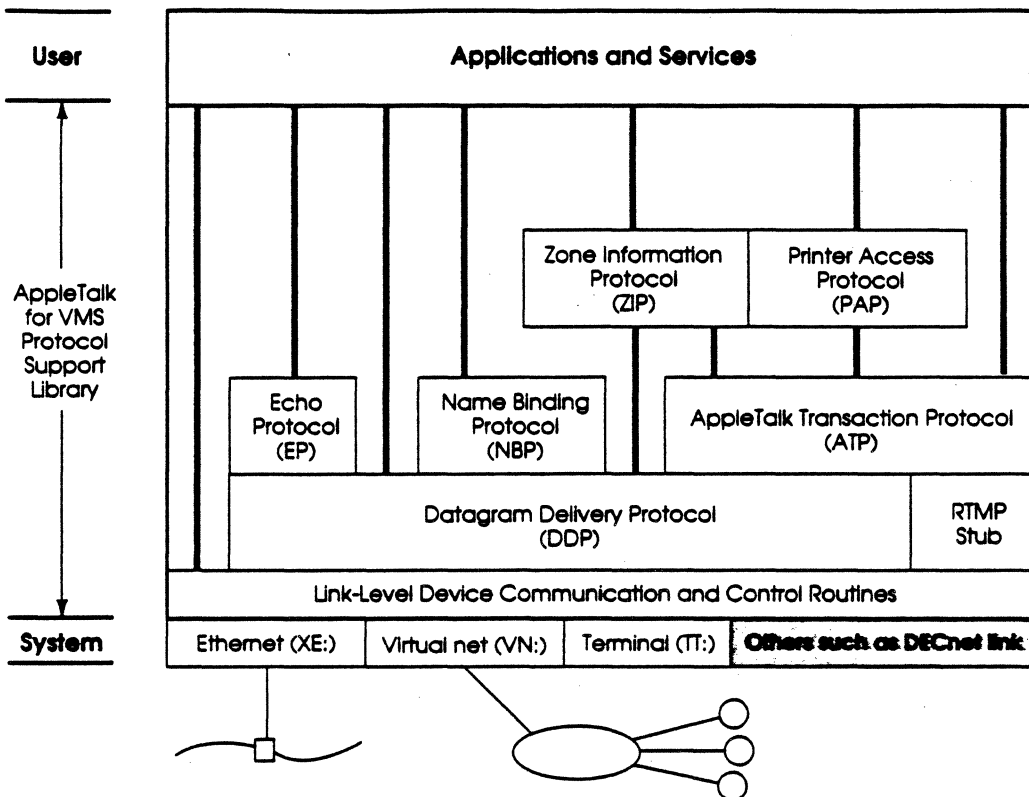


Figure 1-1
Organization of the Protocol Library

How to call Protocol Library procedures

The VAX Procedure Calling and Condition Handling Standard, described in the *Introduction to VAX/VMS System Routines*, defines the interface between your program and the Protocol Library procedures.

To conform to this standard, all the VAX native-mode compiled languages provide mechanisms for coding calls to library procedures. This section briefly outlines how to call AppleTalk for VMS procedures under VAX/VMS. For more information, refer to the *VAX/VMS Run-Time Library Routines Reference Manual*, Part I, Section 2.

When you include a call to a Protocol Library procedure in your program, you must furnish whatever arguments the procedure requires. In most cases, control is returned to your program when the procedure is completed. If the procedure returns a status code, your program must check the status and take appropriate action, depending on whether the procedure was completed successfully or not.

Part II of this manual contains descriptions of each user-accessible procedure in the Protocol Library. The format of these descriptions is based on the de facto Digital standard "routine template" for documentation, described in the *VAX/VMS Run-Time Library Routines Reference Manual*, Part I, Section 1.4.

Digital Equipment Corporation documentation describes calling formats for the more traditional languages such as FORTRAN, COBOL, Pascal, and PL/I. However, the C language has become more commonly used in communications applications, and this manual will therefore use the C language for examples, as well as MACRO-32 where appropriate.

Call formats

Each Protocol Library procedure requires a specific calling sequence. The calling sequence indicates the elements you must include in the language statement that invokes the procedure, and the order of those elements. The following illustrates a call in C and MACRO format:

C: stat = atk\$ddp_opn_skt (&socket);

MACRO: MOVAL SOCKET, -(SP)
 CALLS #1, G^ATK\$DDP_OPN_SKT

- ❖ **Warning:** The C routine call is not capitalized, typical of C programming practices, where capitalized symbols refer to macros and text substitutions handled by the preprocessor via `#define` statements. The descriptions of the routines given in Part II, however, will use capital letters, since this is the Digital convention.

Naming conventions

The Protocol Library entry points follow the VAX conventions for naming global symbols, except that the "Digital" or "system" name form is used. Each global entry point takes the form of:

ATK\$symbol

where ATK is the facility name for the Protocol Library and symbol is the specific entry point name.

All Protocol Library procedures are functions; they either return a "real" result, or they return a status or condition value. Procedures that return a function value signal their errors (see "Errors From Protocol Library Procedures" later in this chapter).

Status and condition codes returned by the Protocol Library fall into two general categories: those that are generated within the Protocol Library, and those that are the result of a call by the Protocol Library to a VAX/VMS system service or run-time library. Status and condition codes generated within the Protocol Library have the following general symbol format (note the double-width underscore):

ATK__ABCMNOXYZ

where ABC, MNO, and XYZ are the first three letters of up to three words that describe the error. For example, "invalid socket number" might be represented by INVSKTNUM. The values for these symbols are defined as global symbols in the library, making it possible to use individual names using the C `#globalvalue` statement. Other languages have similar methods for accessing the value of a globally defined symbol.

Other status and condition codes returned by the Protocol Library may actually be the result of a call to a VAX/VMS system service or run-time library. In that case, symbols are defined in the appropriate "include" file or library supplied by Digital.

Argument passing

A calling program passes an argument list of longwords to a called procedure; each longword specifies a single argument. The called procedure interprets each argument using one of three standard passing mechanisms: by value, by reference, or by descriptor. See the *VAX/VMS Run-Time Library Routines Reference Manual* for argument passing information for traditional languages.

When using the VAX-11 C language, you should be aware that it is capable of passing structured parameters by *value*. If you specify the name of a structure without the address operator "&", a copy of the entire structure is passed to the called procedure. This parameter passing mechanism is not supported by the VAX/VMS Procedure Calling and Condition Handling Standard.

Most high-level languages pass arguments by *reference* or by *descriptor*. Scalar data and structures specific to the application are passed by reference. Data structures commonly supported under VMS have descriptor formats assigned and are passed by descriptor.

The Protocol Library adheres to standards specified by Digital, and does not support null-terminated ("C") strings, nor structures passed by value. Refer to the appropriate language manual for language-specific information regarding argument passing.

Passing arguments by value

Unlike other VAX/VMS languages, the C language normally passes scalar data arguments by value. The uninterpreted 32-bit longword value is pushed onto the stack. For scalars that occupy 16 bits and 8 bits, the argument is loaded into a 32-bit longword, right justified (toward the least significant bits). For example, in C:

```
int a;  
  
func(a);    /* Pass 'a' by value */
```

As stated above, do not pass structured data by value to the Protocol Library.

Passing arguments by reference

This is the natural scalar argument passing mechanism for most VAX/VMS languages. The argument contains the *address* of the data that is the actual argument. The C language uses this mechanism when the name of an array is used without an index, and when the "&" operator precedes the argument name. For example, in C:

```
struct foo stuff;  
  
funct(&stuff);    /* Pass 'stuff' by reference */
```

Passing arguments by descriptor

To facilitate passing information between programs written in different languages, VAX/VMS defines a **descriptor** as a small, tagged, fixed-size data structure that contains descriptive information about (usually) larger, variable-length data structures. When a procedure requires that an argument be passed by descriptor, the argument value must be the address of a descriptor for the data to be passed. Descriptors are an essential part of the VMS system architecture, as many structured data types are used throughout the VMS system, commonly in applications written in a variety of languages.

Some VAX/VMS high-level languages (such as BASIC) use descriptors by default to pass references to structured arguments such as arrays, strings of various types, bit arrays, and so on. The VAX-11 C language has no such facility, so the C programmer must manually construct the descriptor and fill it in with the proper data before passing its address as an argument value. The `DESCRIP.H` header file contains definitions for the legal descriptor types (`DESCRIP.H` is located in `SYSS$LIBRARY:` or in `SYSS$LIBRARY:VAXCDEF.TLB`, depending on how VAX-11 C was installed on your system). Often, a called procedure will check that a descriptor is of the proper type before using the data that it describes.

For more information about using descriptors in C, refer to the Digital manual *Programming in VAX C*.

Event flag usage

The Protocol Library makes use of VAX/VMS event flags to synchronize its internal operations. To avoid conflict with external procedures, the Protocol Library reserves all event flags it uses via the VMS Run-Time Library procedure `LIB$GET_EF`.

Any process using the library should obtain event flags only via the `LIB$GET_EF` call, which coordinates the allocation of event flags among procedures within the same process. This will ensure the proper operation of both the Protocol Library and its client processes.

Dynamic memory allocation

The Protocol Library uses dynamic memory as needed to respond to internal events and client requests. Dynamic memory allocation is accomplished through the VMS Run-Time Library routine, `LIB$GET_VM`.

At this time, other forms of memory allocation are incompatible, including those provided by VMS languages and other run-time library routines.

To avoid conflict, any process using the library should obtain all dynamic memory via the `LIB$GET_VM` call, which coordinates the allocation of memory within a process. Likewise, the `LIB$FREE_VM` call must be used to free memory.

Errors from Protocol Library procedures

Protocol Library procedures indicate error conditions in two ways: by function return and by signaling. Virtually all errors generated in the AppleTalk for VMS Protocol Library are returned by the procedure. The Protocol Library routines do not themselves signal errors except in very unusual circumstances. If a library procedure signals, it is normally a result of an internal error or a bad parameter value (or address) that resulted in a fatal condition during library execution.

Your program may handle signaled errors by establishing a condition handler that is to receive control when an error occurs (see the *VAX/VMS Run-Time Library Routines Reference Manual*). Often, the condition handler must do some clean-up work as well as take the direct action required by the error.

Linking with the Protocol Library

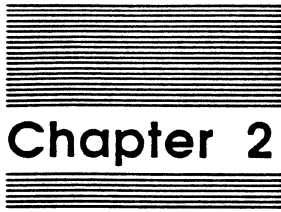
Protocol Library procedures execute entirely in the mode of the caller and are intended to work in VAX/VMS user mode. This section explains how to link your program and Protocol Library procedures together into a single executable unit.

When you link your program, the VAX linker creates an executable image. If your program includes explicit or implicit calls to Protocol Library procedures, the linker automatically searches the system default shareable image library, `IMAGELIB.OLB`, and then searches the system default object module library, `STARLET.OLB`.

Since the Protocol Library routines are *not* contained in either of these libraries, you must explicitly direct the linker to search the Protocol Library shareable image `ATKRTL.EXE`. For information on linker command syntax and options file formats, see the *VAX/VMS Linker Reference Manual*.

When your program calls a Protocol Library procedure, the linker does not copy the procedure into your program's executable image. Instead, a pointer is used to locate the procedure in the shareable image.

Read/write data areas referenced by shareable procedures are, however, copied from the shareable image into your image's virtual address space and mapped, so that the shareable procedures have a separate copy for each calling image. Examples of such read/write data are socket tables, write-data structures, and data buffers used privately by the Protocol Library routines.



Chapter 2

Low-Level I/O Facilities

The AppleTalk for VMS Protocol Support Library employs a low-level I/O mechanism that is kept hidden from the programmer under normal circumstances. However, an understanding of the concepts of buffer management is essential to programmers interested in developing efficient network applications.

The most difficult aspect of network protocol implementation is the efficient handling of incoming data. Data may arrive at any time, so a receiving buffer mechanism must exist. Since an incoming datagram contains not only data but addressing information as well, the routing of the incoming data is only possible once the datagram address has been read. However, the data itself must be stored somewhere while this address is being read.

The simple solution is to read the data from the link into a holding buffer, determine where it is to go next, and then copy the data to the proper location. At the DDP level, a DDP socket client might issue a read request, specifying the address of a buffer that is to contain the next incoming datagram addressed to that socket on that node. The DDP layer in the node must look at the datagram to see what socket it is addressed to, then copy the datagram to that socket's receive buffer.

Under VAX/VMS, the Ethernet DEUNA/DEQNA driver maintains a private pool of buffers to receive incoming Ethernet frames. Then it decides which protocol handler is to receive the frame, and copies it to the handler's receive buffer. If the client process receiving the data copies it one or more times itself, efficiency is degraded.

To maximize efficiency, the Protocol Library contains a set of routines that manage a pool of datagram buffers for use by both the library and the user program. Rather than copy datagrams from one buffer to the next as the data travels through the protocol layers, each datagram is read into or constructed in a buffer, and the address of that buffer is passed from one layer of the protocol to the next.

Link-level communication and the buffer pool

The Protocol Library maintains a pool of datagram buffers that are used to manipulate all datagrams between protocol layers. Library procedures that pass data buffers as arguments and/or return datagrams pass the address of a buffer containing the datagram.

Buffers are not statically allocated by protocol clients; rather, they are obtained from a list of preallocated buffers through calls to buffer management procedures in the Protocol Library. When the library client image prepares to use the library, it must first call the `ATK$INIT_ATALK` procedure, and indicate the number of buffers to be allocated in the pool. Thereafter, all incoming datagrams are read into a free buffer, analyzed, and then passed *by reference* to the appropriate socket client. Outgoing datagrams are built into a buffer, then passed to a transmit routine that returns the buffer to the pool after transmission is complete.

Because your program is ultimately a socket client, it too must manipulate buffers from the pool. This section describes the library's use of the buffer pool, and gives some guidelines on how to best use this facility from your program's code.

- ❖ *Note:* Each process has its own buffer pool. Prior to using other library routines, the process must call `ATK$INIT_ATALK` to establish the allocation of buffers in the pool via a parameter to the call.

Incoming datagrams

After library initialization with the `ATK$INIT_ATALK` call, the datagram buffer pool contains a circularly linked list of empty buffers. Datagrams can start arriving as soon as the data link is opened. Since there is no way to control when a datagram may arrive at the node, the low-level data link input routines attempt to keep an asynchronous read request to the link open at all times. Each such read request specifies a buffer from the pool.

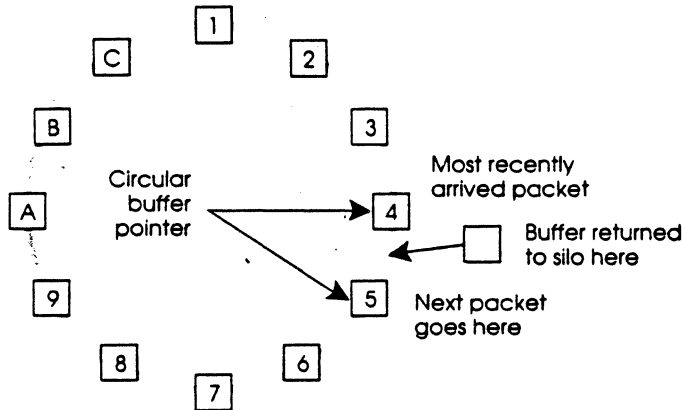


Figure 2-1
Using a circular buffer list as a silo (FIFO)

When a client calls the `ATK$DDP_OPN_SKT` procedure to open a DDP socket, the client normally specifies the address of a socket listener procedure. If there is a socket listener assigned to a socket when a datagram addressed to that socket arrives, the datagram buffer is unlinked from the original circular list, called the **silo list**, and the socket listener procedure is called, with the datagram address passed as a parameter.

If there is no listener assigned to the socket, but the socket is open, the datagram is moved to the *end* of the silo list and is retained for up to 2 seconds. If the socket client issues a read request sometime later, the datagram is then unlinked and passed to the client as before.

In either case, when the socket client has finished with the buffer, it calls a library procedure that returns the buffer to the *beginning* of the silo list (where it will be reused as soon as possible).

If a packet arrives with the address of a nonexistent socket, it is ignored. The current buffer is used for the next incoming packet.

At times, the incoming traffic arrival rate may exceed the rate at which socket clients consume the data. For short periods of time, the silo will absorb such fluctuations by filling empty buffers. Current transport layer protocols in AppleTalk (for example, ATP) provide flow control, so such mismatches should be transient.

Nonetheless, the silo list may become full of incoming datagrams. At this point, when the next datagram arrives, the oldest buffer is reused and the datagram that was stored in it is lost. In well-designed applications this should be a rare occurrence, but in any case it will not be fatal to communication, since the AppleTalk protocols are designed to deal with occasional datagram loss.

Sending datagrams

The datagram buffers are also used in the process of sending datagrams. Clients wishing to send a datagram must allocate a buffer from the silo list, and fill in the data and the DDP header, including destination internet address and source socket number. The client then calls the `ATK$DDP_WRITE` procedure to route and transmit the datagram. First, the routing process fills in the source node and network numbers (except for transient packets being handled by an internet router process).

If the datagram is sent without specifying a user completion routine (whether asynchronous or not), the library handles silo buffer management automatically. The buffer is returned to the silo when the transmit is complete. If the client specifies a completion routine, the client is responsible for returning the buffer to the silo.

If the local client issues a large number of transmit requests, the silo might become depleted, containing only the buffer currently assigned for receive. However, the natural flow control of client protocols such as ATP reduces this risk. Ideally, buffer sizes specified at the time of program design should anticipate application needs.

Tuning buffer allocation

When a program prepares to use the Protocol Library, it calls `ATK$INIT_ATALK`. At that time, you may specify the number of datagram buffers to be allocated to your process. If you allocate too few, datagram loss will be excessive, your application's throughput will be poor, and network overhead will be high due to excessive retries. If you allocate too many, your application will be using excessive virtual memory. The objective is for the silo list to contain enough buffers to absorb short-term fluctuations in datagram flow.

Note that one buffer is allocated permanently to a port when it is opened. This buffer is used as a "last chance" place to put incoming packets if the silo becomes empty. Packets so received are not available to the application and are lost.

Devices and ports

The Protocol Library has the support needed to implement internetwork router processes (bridges). At the link level, what distinguishes a router from an end node is that the router is connected to more than one network.

A connection to a network is referred to as a **port**. A nonbridge has a single port. A bridge has two or more ports, and is capable of routing DDP datagrams between them.

A port on a VMS system can be composed of some or all of the following:

- ☐ an **I/O channel** to a communication device (Ethernet, VLAP, and so on)
- ☐ a **protocol type**, if supported by the link layer (if applicable)
- ☐ the **physical address** of the port as seen by others on its LAP
- ☐ the **AppleTalk network number** of the net to which the device is attached (DDP networks only)

- the **AppleTalk node number** by which the system is known on that net (DDP networks only)

The last two items are valid only for DDP-speaking networks such as ALAP and VLAP. They would be set equal to zero for non-DDP networks, and for point-to-point links.

Not all link-level networks support protocol types and/or network numbers. For example, the VLAP virtual network does not distinguish between protocol types, but both Ethernet and ALAP do have link-level protocol types.

Port drivers

The library uses a common interface to communicate with the various port types, and the port-specific routines for each type are kept in separate images called **port drivers**. The library image (ATKRTL.EXE) contains no port-level code. Each port is supported by a separate VMS image file, the port driver, which contains the routines needed to start, stop, and communicate via that port. The next section describes the devices and ports supported by the current release. Detailed information on individual port drivers may be found in Appendix A.

The library client process establishes communication by issuing two Protocol Library calls, `ATK$INIT_PORT` and `ATK$OPEN_PORT`. `ATK$INIT_PORT` loads the port driver into the process's address space and calls the driver at its initialization entry point. This is a one-time initialization done only at load time. `ATK$OPEN_PORT` starts port driver operations, performs device-specific startup, and sets up port descriptor data structures.

The library loads the driver by calling the VMS Run-Time Library procedure `LIB$FIND_IMAGE_SYMBOL`, which activates the port driver image and maps it into the process virtual address space. It then calls the driver at its initialization entry point. The driver performs device-specific startup and fills in some of the port descriptor fields, including the driver's read, write, and shutdown routine entry points.

Since the port drivers are separate from the library itself, it is possible for a user to write his own port driver. The driver must be linked to make it a shareable image, and placed in `SYSS$SHARE` (the default for `LIB$FIND_IMAGE_SYMBOL`). No changes are needed to the library itself. To use the new port driver, call `ATK$OPEN_PORT` with its image file name and device-specific string. For more information on how to develop a port driver, contact Apple Computer, Inc.

Device types

Port drivers are provided for each communication **device type** supported by the library. Ports fall into two general classes, those that support DDP addressing and those that do not. DDP-speaking ports may be used by end-node processes and bridges alike. Non-DDP ports are useful only to bridges, for communicating with other bridges.

The current release of the library provides drivers for the device types shown in Table 2-1.

Table 2-1
Supported communication devices

DEV	Driver*	DDP	Description
VN:	VLAP	yes	The virtual AppleTalk device supplied by Apple. This is a software-only network simulator that provides interprocess communication transparently, creating the illusion that each process is connected to a real DDP network.
XE: XQ:	ELAP	no	The Ethernet device supplied by Digital. On Q-bus systems, the controller is called the DEQNA; on Unibus systems, it is called the DEUNA or DELUA. DDP is not supported on Ethernet ports.
TT:	AHLAP	no	Asynchronous half-bridge port. Uses AppleTalk asynchronous protocol. Useable on most VMS serial asynchronous devices.
	DLAP	no	DECnet logical links, used as point-to-point circuits connecting router processes on remote nodes, providing long-haul internetwork communication using technologies such as satellite and X.25 packet networks.

* Driver image file names are of the form ATKxxxx.EXE, where "xxxx" is the driver name shown in the table. Normally, the images are kept in SYS\$SHARE.

Communicating on ports

Low-level I/O is performed by calling library routines, and specifying both the port ID of the desired port and the physical address of the destination on the corresponding data link. Note that the physical address is specific to the port's LAP, and that some ports do not support DDP addressing.

Before low-level I/O can be performed, your program must call the `ATK$INIT_PORT` and `ATK$OPEN_PORT` routines, which set up the port data structure, start I/O on the specified device with the appropriate protocol type, and return a port ID.

Unless your application is a bridge, you must open only one port. Most VMS processes will communicate via the VLAP virtual network port, through the bridge process to the outside world.

Virtual network (VLAP)

The VAX/VMS Virtual AppleTalk Driver supplied by Apple provides local network services between processes on a single VAX/VMS system. Both direct addressing and broadcasting are supported.

Ethernet (ELAP)

At present, the Ethernet link-level protocol can be implemented only as a "backbone" network used to carry DDP datagrams between AppleTalk internet routers. The ELAP port can only be opened and used by AppleTalk for VMS bridges.

The physical address for Ethernet is a 48-bit node number. These node numbers are usually statically assigned, either from an address burned into the controller's ROM, or if DECnet is in use, by the DECnet node executor.

Asynchronous half-bridge (AHLAP)

This is a non-DDP port used to provide communication between bridges using asynchronous protocol. This port is compatible with the Hayes InterBridge™ link.

DECnet (DLAP)

The DLAP port driver enables communication between the AppleTalk for VMS Protocol Library and a port connected to a DECnet link.





Part II



Protocol Library Routines

This part of the *Protocol Support Library Reference Manual* provides a description of each AppleTalk for VMS Protocol Library routine.

Routine descriptions are listed in alphabetical order by the name of each routine call. Listing formats are based on the documentation convention set forth in the *VAX/VMS Run-Time Library Routines Reference Manual* (Digital Equipment Corporation).

- ❖ *Reminder:* C-language routine calls are not capitalized, according to C programming practices, where capitalized symbols refer to macros and text substitutions handled by the preprocessor via `#define` statements. The descriptions of the routines given here, however, will use capital letters, since this is the Digital convention.

ATK\$ATP_CANCEL_REQ – Cancel ATP request

This routine cancels an outstanding ATP send request. The cancelled request completes with an abort.

Format `ATK$ATP_CANCEL_REQ      transaction-id`

Returns type: longword (unsigned)
 access: write only
 mechanism: by value

One of the values listed below under "Conditions."

Arguments `transaction-id`

type: word (unsigned)
access: read only
mechanism: by value

The unique 16-bit identifier assigned to the transaction. The `transaction-id` argument contains the value of the transaction ID assigned by the `ATP_SND_REQ` routine.

Conditions `SS$_NORMAL` Routine successfully completed.

 `ATK__NOTRFND` No matching transaction found.



ATK\$ATP_CLS_SKT – Close an ATP socket

This routine closes an ATP responding socket. Any outstanding get requests or send responses are completed with an abort.

Format ATK\$ATP_CLS_SKT socket

Returns type: longword (unsigned)
 access: write only
 mechanism: by value

One of the values listed below under “Conditions.”

Arguments **socket**
 type: byte (unsigned)
 access: read only
 mechanism: by value

The number of the socket to be closed. **Socket** is the value of the 1-byte ATP socket number.

Conditions \$\$\$_NORMAL Routine successfully completed.

Any condition values returned by the following AppleTalk Protocol Library routines:

ATK\$DDP_CLS_SKT

ATK\$ATP_GET_REQ - Receive an ATP request

This routine establishes the context for receiving an incoming ATP request on a previously opened ATP responding socket.

Format

ATK\$ATP_GET_REQ	socket
	, net-address
	, request-buffer
	, request-buf-size
	, request-length
	, user-bytes
	, xo-mode
	, transaction-id
	, bitmap
	, async-flag
	, completion-routine
	, completion-parameter

Returns

type:	longword (unsigned)
access:	write only
mechanism:	by value

One of the values listed below under "Conditions."

Arguments

socket	
type:	byte (unsigned)
access:	read only
mechanism:	by value

The number of the socket on which to receive the request. The **socket** argument contains the value of the socket number. The socket must have been previously opened as an ATP-responding socket via a call to ATK\$ATP_OPN_SKT.

net-address

type:	unspecified
access:	write only
mechanism:	by reference

The 4-byte internet address of the requester. The **net-address** argument is the address of the internet address block (see "Description" below). The first 2 bytes contain the 16-bit network address. The first byte is the least significant byte, followed by the most significant byte. The third byte contains the node number, and the fourth contains the socket number.

request-buffer

type: unspecified
access: read only
mechanism: by reference

The buffer to contain the request data received with the ATP request. The `request-buffer` argument is the address of a buffer that will receive the incoming request packet.

request-buf-size

type: word (unsigned)
access: read only
mechanism: by value

The size in bytes of the buffer supplied to receive the request data. The `request-buf-size` argument contains the value of the size of the request buffer.

request-length

type: word (unsigned)
access: read only
mechanism: by reference

The size in bytes of the received request. The `request-length` argument contains the address of a word to be filled in with the size of the request data.

user-bytes

type: unspecified
access: write only
mechanism: by reference

The 4 user bytes received in the ATP header of the request packet. The `user-bytes` argument is the address of a 4-byte area that receives the user bytes. A zero may be specified if no user bytes are desired.

xo-mode

type: byte (unsigned)
access: write only
mechanism: by reference

A flag value indicating whether or not the transaction was requested in the ATP exactly once mode. The `xo-mode` argument contains the address of a byte to receive the flag. Any nonzero value indicates exactly once mode.

transaction-id

type: word (unsigned)
access: write only
mechanism: by reference

The unique 16-bit identifier assigned to the transaction. The `transaction-id` argument contains the address of the word that receives the transaction ID.

bitmap

type: byte (unsigned)
access: write only
mechanism: by reference

The **bitmap** parameter is the address of a byte that receives the requester's buffer descriptor bitmap.

async-flag

type: byte (unsigned)
access: read only
mechanism: by value

A flag value indicating whether or not the get request is to be issued asynchronously; that is, whether the **ATK\$ATP_GET_REQ** procedure returns control to the caller as soon as the get request has been issued. The **async-flag** argument contains the value of the flag. Any nonzero value indicates asynchronous mode.

completion-routine

type: procedure entry mask
access: call without stack unwinding
mechanism: by reference

The entry mask of a procedure that is called when a request arrives. The **completion-routine** argument contains the address of this procedure entry mask.

This routine is called with the following format:

Format	completion-routine	completion-parameter , completion-status
---------------	---------------------------	---

Arguments	completion-parameter
------------------	-----------------------------

type:	longword
access:	read only
mechanism:	by value

	completion-status
--	--------------------------

type:	longword
access:	read only
mechanism:	by value

The library calls the completion routine with the **completion-parameter** and the **completion-status** of the get request. A zero may be specified if no completion routine is desired.

completion-parameter

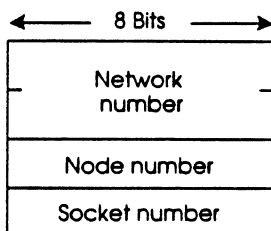
type: longword
access: read only
mechanism: by value

A caller-supplied value that is passed as an argument to the completion routine. The **completion-parameter** argument is a longword containing the caller-supplied value.

Description

This routine is used to prepare for receiving an ATP request on a particular socket. The socket must have been opened through a call to `ATK$ATP_OPN_SKT`.

One of the parameters specifies the address of an internet address block that receives the internet address of the requester. The internet address block contains an internet address as follows:



For more information, consult *Inside AppleTalk*.

Conditions

<code>SS\$_NORMAL</code>	Routine successfully completed.
<code>ATK__SKTNOTOPN</code>	Socket was not open.
<code>ATK__BUFOVRFLW</code>	Request buffer was too small for request data.
<code>ATK__GETREQABO</code>	Get Request call was aborted by a call to <code>ATK\$ATP_CLS_SKT</code> .

Any condition values returned by the following VMS Run-Time Library routines:

`LIB$GET_VM`

ATK\$ATP_OPN_SKT – Open an ATP socket

This routine opens an ATP responding socket, creating a network-visible entity.

Format ATK\$ATP_OPN_SKT socket

Returns type: longword (unsigned)
 access: write only
 mechanism: by value

One of the values listed below under "Conditions."

Arguments **socket**

 type: byte (unsigned)
 access: modify
 mechanism: by reference

The requested ATP socket number, or zero. *Socket* is the address of the 1-byte ATP socket number. If the contents are set in the range 1...127 when the call is issued, the requested *well-known socket* is opened. If the contents are set to zero prior to calling, the procedure opens a *dynamic socket* and places the assigned socket number into the *socket* prior to returning.

Conditions SSS_NORMAL Routine successfully completed.

Any condition values returned by the following AppleTalk Protocol Library routines:

ATK\$DDP_OPN_SKT

Any condition values returned by the following VMS Run-Time Library routines:

LIB\$GET_VM

ATK\$ATP_SND_REQ – Send an ATP request

This routine initiates an ATP transaction by establishing the transaction context and sending an ATP request to the intended responder.

Format

ATK\$ATP_SND_REQ **net-address**
 , request-buffer
 , request-length
 , buffer-descriptors
 , bitmap
 , num-of-responses
 , user-bytes
 , xo-mode
 , transaction-id
 , async-flag
 , completion-routine
 , completion-parameter
 [, retry-count
 [, retry-interval]]

Returns

type: longword (unsigned)
access: write only
mechanism: by value

One of the values listed below under "Conditions."

Arguments

net-address

type: unspecified
access: read only
mechanism: by reference

The 4-byte internet address of the responder. The **net-address** argument is the address of the responder's internet address block (see "Description" below). The first 2 bytes contain the 16-bit network address. The first byte is the least significant byte, followed by the most significant byte. The third byte contains the node number, and the fourth contains the socket number.

request-buffer

type: unspecified
access: read only
mechanism: by reference

The request data to be included in the ATP request. The **request-buffer** argument is the address of a buffer containing the data.

request-length

type: word (unsigned)
access: read only
mechanism: by value

The size, in bytes (up to 578), of the request data. The `request-length` argument contains the value of the request data size. A zero may be specified if the requester is only interested in sending the user-bytes.

buffer-descriptors

type: unspecified
access: read only
mechanism: by reference

A list of buffer descriptors that describe the set of buffers to receive the response data (see "Description" below). The `buffer-descriptors` argument is the address of the first buffer descriptor in the list. A zero may be specified if the requester is not expecting a response and the responder does not expect a transaction release.

bitmap

type: byte (unsigned)
access: read only
mechanism: by reference

The `bitmap` parameter is the address of a byte describing which buffer descriptor elements should be used for the response.

num-of-responses

type: byte (unsigned)
access: write only
mechanism: by reference

The number of buffers (up to eight) actually received in the reply. The `num-of-responses` argument contains the address of a byte to receive the number. A zero may be specified if the requester is not interested in the number of response buffers received.

user-bytes

type: unspecified
access: read only
mechanism: by reference

The 4 user bytes to be placed into the ATP header of the request packet. The `user-bytes` argument is the address of a 4-byte area containing the user bytes. A zero may be specified if no user bytes are desired.

xo-mode

type: byte (unsigned)
access: read only
mechanism: by value

A flag value indicating whether or not the transaction is to be performed in the ATP exactly once mode. The xo-mode argument contains the value of the flag. Any nonzero value indicates exactly once mode.

transaction-id

type: word (unsigned)
access: write only
mechanism: by reference

The unique 16-bit identifier assigned to the transaction. The transaction-id argument contains the address of the word to receive the transaction ID. A zero may be specified if the requester is not interested in knowing the transaction ID.

async-flag

type: byte (unsigned)
access: read only
mechanism: by value

A flag value indicating whether or not the transaction is to be issued asynchronously; that is, whether the ATK\$ATP_SND_REQ procedure returns control to the caller as soon as the transaction request has been issued. The async-flag argument contains the value of the flag. Any nonzero value indicates asynchronous mode.

completion-routine

type: procedure entry mask
access: call without stack unwinding
mechanism: by reference

The entry mask of a procedure that is called when the transaction completes. The completion-routine argument contains the address of this procedure entry mask.

This routine is called with the following format:

Format	completion-routine	completion-parameter , completion-status
---------------	--------------------	---

Arguments	completion-parameter
------------------	----------------------

type:	longword
access:	read only
mechanism:	by value

completion-status

type:	longword
access:	read only
mechanism:	by value

The library calls the completion routine with the completion-parameter and the completion-status of the send request. The completion routine must do a normal return when it completes. A zero may be specified if no completion routine is desired.

completion-parameter

type: longword
access: read only
mechanism: by value

A caller-supplied value that is passed as an argument to the completion routine. The completion-parameter argument is a longword containing the caller-supplied value.

retry-count

type: word (unsigned)
access: read only
mechanism: by value

The number of times to retry the request. The retry-count argument contains the value of the retry count. A value of 65535 (FFFF hexadecimal) causes infinite retries. If retry-count is not specified, a default value of 8 is used.

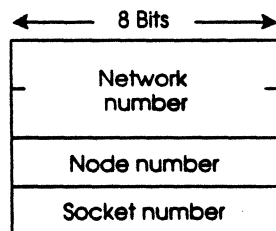
retry-interval

type: word (unsigned)
access: read only
mechanism: by value

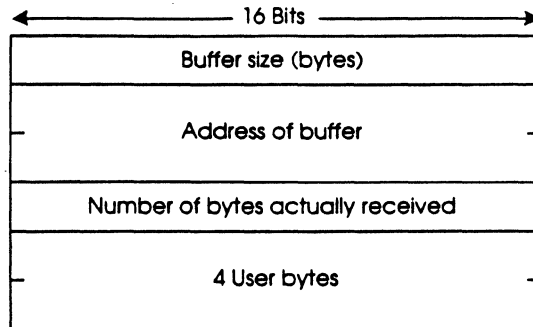
The interval, in seconds, between request retries. The retry-interval argument contains the value of the retry interval. If retry-interval is not specified or is zero, a default value of 2 is used.

Description

This routine performs a transaction according to the rules of the AppleTalk Transaction Protocol. Two special data structures are needed: an internet address block, and one or more buffer descriptors. The internet address block contains an internet address as follows:



A buffer descriptor (BDS) is used for each buffer allocated for the response. The buffer descriptor has the following structure:



The caller must set up a BDS for each buffer expected in the response. The buffer descriptors must be physically contiguous in memory, and the `buffer-descriptors` argument set to point to the first BDS in the list.

For more information, consult *Inside AppleTalk*.

Conditions

<code>SS\$ _NORMAL</code>	Routine successfully completed.
<code>ATK__TOOMNYRSP</code>	Too many response buffers. ATP allows for a maximum of eight response buffers in a single transaction.
<code>ATK__REQTOOBIG</code>	Request buffer is too big. ATP allows for a maximum request buffer size of 578 bytes.
<code>ATK__BUFOVRFLW</code>	One of the responses is larger than its corresponding buffer described in the BDS.
<code>ATK__REQABO</code>	Send request was aborted by a call to <code>ATK\$ATP_CANCEL_REQ</code> .
<code>ATK__RTYCNTEXC</code>	No response was received before the retry count was exceeded.

Any condition values returned by the following AppleTalk Protocol Library routines:

`ATK$GET_DATAGRAM`

`ATK$DDP_OPN_SKT`

`ATK$DDP_WRITE`

`ATK$TMRINS`

Any condition values returned by the following VMS Run-Time Library routines:

`LIB$GET_VM`

ATK\$ATP_SND_RSP – Send an ATP response

This routine initiates an ATP response. If the transaction is "exactly once," it doesn't complete until a release is received from the requester or the release timer expires (30 seconds).

Format

ATK\$ATP_SND_RSP	socket
	,net-address
	,buffer-descriptors
	,bitmap
	,user-bytes
	,xo-mode
	,transaction-id
	,async-flag
	,completion-routine
	,completion-parameter

Returns

type:	longword (unsigned)
access:	write only
mechanism:	by value

One of the values listed below under "Conditions."

Arguments

socket	
type:	byte (unsigned)
access:	read only
mechanism:	by value

The number of the socket on which the response is to be sent. The **socket** argument contains the value of the socket number. The socket must have been previously opened as an ATP-responding socket via a call to ATK\$ATP_OPN_SKT.

net-address

type:	unspecified
access:	read only
mechanism:	by reference

The 4-byte internet address of the requester. The **net-address** argument is the address of the requester's internet address block (see "Description" below). The first 2 bytes contain the 16-bit network address. The first byte is the least significant byte, followed by the most significant byte. The third byte contains the node number, and the fourth contains the socket number.

buffer-descriptors

type: unspecified
access: read only
mechanism: by reference

A list of buffer descriptors that describe the set of buffers containing the response data (see "Description" below). The **buffer-descriptors** argument is the address of the first buffer descriptor in the list.

bitmap

type: byte (unsigned)
access: read only
mechanism: by reference

The **bitmap** parameter is the address of a byte describing which buffer descriptor elements should be sent as part of the response.

user-bytes

type: unspecified
access: write only
mechanism: by reference

The 4 user bytes that are received with the release packet in an exactly once transaction. The **user-bytes** argument contains the address of the 4-byte area that receives the user bytes. A zero may be specified if you do not wish to receive the release packet's user bytes.

xo-mode

type: byte (unsigned)
access: read only
mechanism: by value

A flag value indicating whether or not the response is part of an exactly once transaction. The **xo-mode** argument contains the value of the flag. Any nonzero value indicates exactly once mode.

transaction-id

type: word (unsigned)
access: read only
mechanism: by value

The unique 16-bit identifier assigned to the transaction. The **transaction-id** argument contains the value of the transaction ID.

async-flag

type: byte (unsigned)
access: read only
mechanism: by value

A flag value indicating whether or not the response is to be issued asynchronously; that is, whether the ATK\$ATP_SND_RSP procedure returns control to the caller as soon as the response has been issued. The `async-flag` argument contains the value of the flag. Any nonzero value indicates asynchronous mode. An asynchronous send response is only meaningful when the response is part of an exactly once transaction.

completion-routine

type: procedure entry mask
access: call without stack unwinding
mechanism: by reference

The entry mask of a procedure that is called when the send response completes. The `completion-routine` argument contains the address of this procedure entry mask.

This routine is called with the following format:

Format	<code>completion-routine</code>	<code>completion-parameter</code> <code>, completion-status</code>
---------------	---------------------------------	---

Arguments	<code>completion-parameter</code>
------------------	-----------------------------------

type:	longword
access:	read only
mechanism:	by value

	<code>completion-status</code>
--	--------------------------------

type:	longword
access:	read only
mechanism:	by value

The library calls the completion routine with the `completion-parameter` and the `completion-status` of the send response. The completion routine must do a normal return when it completes. A send response completion routine is only meaningful when the response is part of an exactly once transaction. A zero may be specified if no completion routine is desired.

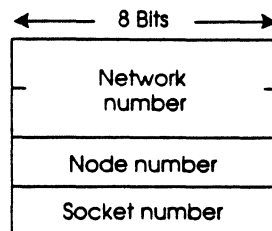
completion-parameter

type: longword
access: read only
mechanism: by value

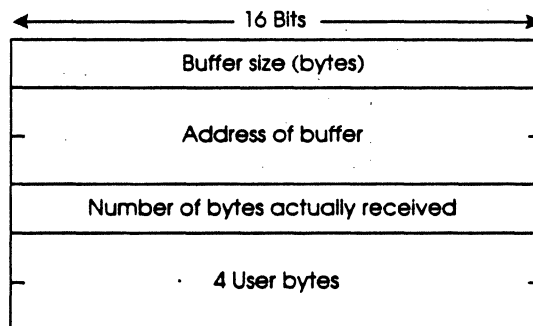
A caller-supplied value that is passed as an argument to the completion routine. The `completion-parameter` argument is a longword containing the caller-supplied value.

Description

This routine sends a response according to the rules of the AppleTalk Transaction Protocol. Two special data structures are needed: an internet address block and one or more buffer descriptors. The internet address block contains an internet address as follows:



A buffer descriptor is used for each buffer allocated for the response. The buffer descriptor has the following structure:



The caller must set up a BDS for each buffer expected in the response. The buffer descriptors must be physically contiguous in memory, and the buffer-descriptors argument set to point to the first BDS in the list.

For more information, consult *Inside AppleTalk*.

Conditions

SS\$ _NORMAL	Routine successfully completed.
ATK__TOOMNYRSP	Too many response buffers. ATP allows for a maximum of eight response buffers in a single transaction.
ATK__NOXOTRN	No exactly once transaction is active. The response was issued with the xo-mode flag set, but there isn't a matching active exactly once transaction.
ATK__RSPTOOBIG	Response buffer is too big. ATP allows for a maximum response buffer size of 578 bytes.
ATK__RSPABO	Send response was aborted by a call to ATK\$ATP_CANCEL_RSP.
ATK__NORELRCV	No transaction release packet received.

Any condition values returned by the following AppleTalk Protocol Library routines:

ATK\$GET_DATAGRAM

ATK\$DDP_WRITE

ATK\$BYTESWAP – Byte swapper

This is a general-purpose byte swapping routine. It byte swaps a stream of bytes according to a byte swap format string.

```
Format      ATK$BYTESWAP      buffer
            ,fmt-str
```

Arguments **buffer**

```

type:      unspecified
access:    modify
mechanism: by reference

```

The buffer on which to perform the byte swapping. The `buffer` argument is the address of the buffer.

fmt-str

```

type:      unspecified
access:    read only
mechanism: by reference

```

The format of the byte swapping that is desired. `Fmt-str` is the address of an array of ASCII characters terminated by a null character (a C-string) that defines the format of the buffer. An ASCII "1" means the corresponding field in the buffer is a byte. An ASCII "2" means the corresponding field is a word, and to byte swap it. An ASCII "4" means the corresponding field is a longword, and to byte swap and word swap it. Only the ASCII characters 1, 2, and 4 are legal.

Description	This routine is provided to help solve what is known as "the byte swap problem." Simply stated, some machines address words and longwords by their most significant byte, others by their least significant byte. The 68000 microprocessor family falls into the former category, the VAX falls into the latter.
--------------------	--

Words and longwords in data passed between systems with different storage conventions must be converted on the way in and out of the host. Since AppleTalk was developed around 68000-family machines, the 68000 convention applies to data transfer on AppleTalk. Under VMS, you are responsible for converting the "data" portion of datagrams. The library handles protocol-related items, such as the length field, in datagrams.

ATK\$BYTESWAP is isomorphic; it works for converting both incoming and outgoing data. An example may help clarify usage:

```
struct myStuff
{
    char a;
    short b;
    long c;
    char e[4];
    long f;
    short g;
};
static char myStuffByteSwap[] = "1241111142";
static struct myStuff foo;
...
    atk$byteswap(&foo, myStuffByteSwap);
```

Note that the conversion is C-string-driven. The string can be stored as a static array next to the declaration of the data structure itself, then called throughout a program. This way, the structure definition and the corresponding transformation string may be easily maintained if changes are needed.

ATK\$CHECKSUM - Compute DDP checksum

This routine computes the checksum of a buffer using the DDP checksum algorithm.

```
Format      ATK$CHECKSUM      buffer
              ,buffer-length
```

Returns

type:	word (unsigned)
access:	write only
mechanism:	by value

Computed DDP checksum.

Arguments	buffer
type:	unspecified
access:	read only
mechanism:	by reference

The buffer on which to compute the DDP checksum. The `buffer` argument is the address of the buffer.

buffer-length

type:	longword
access:	read only
mechanism:	by value

The length, in bytes, of the buffer. The `buffer-length` argument contains the value of the buffer size.

ATK\$CLOSE_PORT – Close a port driver

This routine closes an AppleTalk for VMS port that has previously been opened.

Format ATK\$CLOSE_PORT port-id

Returns type: longword (unsigned)
 access: write only
 mechanism: by value

One of the values listed below under "Conditions."

Arguments port-id

 type: longword (unsigned)
 access: read only
 mechanism: by value

This is the numeric identifier of the port. The port-id argument is the value of the unsigned longword identifier.

Conditions SS\$ _NORMAL Routine successfully completed.

ATK\$DDP_CLS_SKT – Close a DDP socket

This routine closes a DDP socket, removing a network-visible entity.

Format ATK\$DDP_CLS_SKT socket

Returns type: longword (unsigned)
 access: write only
 mechanism: by value

One of the values listed below under "Conditions."

Arguments socket

 type: byte (unsigned)
 access: read only
 mechanism: by value

The number of the socket to be closed. Socket is the value of the 1-byte DDP socket number.

Conditions \$\$\$_NORMAL Routine successfully completed.

 ATK__SKTNOTOPN Socket was not open.

This routine opens a DDP socket, creating a network-visible entity, and assigns a client AST listener procedure.

Returns	type:	longword (unsigned)
	access:	write only
	mechanism:	by value

Arguments **socket**

Socket is the address of the 1-byte DDP socket number. If the value of this argument is set in the range 1...127 when the call is issued, the requested well-known socket is opened. If this argument value is set to zero, the procedure opens a dynamic socket, with a socket number in the range 128...254 (dynamic socket numbers are assigned only by the procedure; you may not specify one of these values when setting the **socket** parameter). If the contents are set to 255 (FF hex), the socket will receive all incoming datagrams whose DDP destination address is not the local node.

```

type:      procedure entry mask
access:    call without stack unwinding
mechanism: by reference

```

Entry mask of a socket listener procedure to call at AST level upon arrival of a packet addressed to the socket. The listen-astadr argument is the address of this procedure entry mask.

This routine is called with the following format:

Arguments	datagram	
	type:	unspecified
	access:	read only
	mechanism:	by reference

port-id
 type: longword (unsigned)
 access: read only
 mechanism: by value

The listener procedure is passed the address of the arriving datagram and the port ID of the port on which the datagram arrived.

Description

This routine opens a DDP socket for communication. Both statically assigned ("well-known") sockets and dynamic sockets are supported. See the description of the socket parameter above.

If the specified socket number is 255 (FF hex), the "socket" is not a true socket. Instead, the client will receive all incoming packets whose DDP destination address is not the local node. This feature is provided to allow implementation of internetwork routers, which must forward all such datagrams.

The socket listener procedure is called at AST level when a packet arrives addressed to that socket. The listener is passed the address of the arriving datagram and the port ID of the port on which the datagram arrived. The socket client is responsible for returning the buffer to the silo when its data is no longer needed, by using the ATK\$RETURN_DATAGRAM routine (see "Incoming Datagrams" in Part I, Chapter 2).

If listen-astadr is not supplied, or if an address of zero (NULL) is given, no listener procedure is assigned to the socket. In this case, the socket client is responsible for periodically calling ATK\$DDP_READ to fetch incoming datagrams from the silo buffer manually. Failure to do so often enough will result in data loss. *Polling for incoming datagrams is not recommended; you should use a socket listener except in unusual cases.*

Conditions

SS\$_NORMAL	Routine successfully completed.
ATK__SKTOPN	Requested well-known socket is already open.
ATK__INVSKTNUM	Invalid supplied socket number. Requested well-known socket number greater than 127.
ATK__NOSKTS	No more sockets to allocate. All dynamic numbers are in use.

```
Format      ATK$DDP_READ      socket
            ,datagram-ptr
```

Returns type: longword (unsigned)
 access: write only
 mechanism: by value

One of the values listed below under "Conditions."

Arguments **socket**

```

type:      byte (unsigned)
access:    read only
mechanism: by value

```

The number of the socket on which to receive a datagram. The `socket` argument contains the value of the socket number. The socket must be open.

datagram_ptr

```
type:      unspecified
access:    write only
mechanism: by reference
```

The oldest DDP datagram from the silo queue addressed to the specified socket. The `datagram_ptr` argument is the address of a DDP datagram pointer that receives the address of the oldest DDP datagram.

Conditions	SS\$ _NORMAL	Routine successfully completed.
	ATK__INVSKTNUM	Invalid or not an open socket number.
	ATK_ NODGMWTG	No datagram waiting for the specified socket.



ATK\$DDP_WRITE – Transmit a DDP datagram

This routine transmits a DDP datagram for routing according to the supplied parameters.

Format

ATK\$DDP_WRITE	datagram , datagram-length , dst-netaddr , src-netaddr , ddp-protocol-type , async-flag , completion-routine , completion-parameter
----------------	--

Returns

type:	longword (unsigned)
access:	write only
mechanism:	by value

One of the values listed below under "Conditions."

Arguments **datagram**

type:	unspecified
access:	modify
mechanism:	by reference

The DDP datagram to be transmitted. The datagram argument contains the address of the DDP datagram.

- ❖ *Warning:* The contents of the datagram may be modified by this procedure. See the descriptive section below.

datagram-length

type:	word (unsigned)
access:	read only
mechanism:	by value

The size, in bytes (up to 586), of the data portion of the datagram. The datagram-length argument contains the value of the datagram data size.

dst-netaddr

type:	unspecified
access:	read only
mechanism:	by reference

The 4-byte destination internet address of the buffer. The dst-netaddr argument is the address of the internet address block (see "Description" below). The first 2 bytes contain the 16-bit network address. The first byte is the least significant byte, followed by the most significant byte. The third byte contains the node number, and the fourth contains the socket number.

src-netaddr

type: unspecified
access: modify
mechanism: by reference

The 4-byte internet address of the sender. The `src-netaddr` argument is the address of the internet address block (see "Description" below). The first 2 bytes contain the 16-bit network address. The first byte is the least significant byte, followed by the most significant byte. The third byte contains the node number, and the fourth contains the socket number.

If the value of `src-netaddr` is a pointer to an internet address block, `ATK$DDP_WRITE` will check the node number of the block for a zero value. If this value is zero, `ATK$DDP_WRITE` will fill in the local node number and network number of the port on which the datagram is to be transmitted. Once the internet address block is complete, `ATK$DDP_WRITE` will copy it to the DDP header of the datagram buffer. If the value of `src-netaddr` is zero, `ATK$DDP_WRITE` will not alter the source address in the DDP buffer (this feature is intended to support internetwork router processes that transmit datagrams originating on remote nodes).

ddp-protocol-type

type: byte (unsigned)
access: read only
mechanism: by value

The DDP protocol type to assign to the datagram. The `ddp-protocol-type` argument is the value of the DDP protocol type.

async-flag

type: byte (unsigned)
access: read only
mechanism: by value

A flag value indicating whether or not the transaction is to be issued asynchronously; that is, whether the `ATK$DDP_WRITE` procedure returns control to the caller as soon as the buffer has been transmitted. The `async-flag` argument contains the value of the flag. Any nonzero value indicates asynchronous mode.

completion-routine

type: procedure entry mask
access: call without stack unwinding
mechanism: by reference

The entry mask of a procedure that is called when the buffer contents have been transmitted. The `completion-routine` argument contains the address of this procedure entry mask.

This routine is called with the following format:

Format	<code>completion-routine</code>	<code>completion-parameter</code> <code>, completion-status</code>
---------------	---------------------------------	---

Arguments

<code>completion-parameter</code>	
type:	longword
access:	read only
mechanism:	by value
<code>completion-status</code>	
type:	longword
access:	read only
mechanism:	by value

The library calls the completion routine with the `completion-parameter` and the `completion-status` of the transaction. The completion routine must do a normal return when it completes. A zero may be specified if no completion routine is desired.

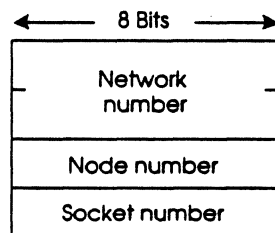
completion-parameter

type:	longword
access:	read only
mechanism:	by value

A caller-supplied value that is passed as an argument to the completion routine. The `completion-parameter` argument is a longword containing the caller-supplied value.

Description

This procedure performs datagram routing and transmission according to the rules of DDP. An internet address block is required for specifying the source and destination internet address of the datagram. The format of the internet address block is as follows:



If the value of `src-netaddr` is a pointer to an internet address block, `ATK$DDP_WRITE` will check the node number of the block for a zero value. If this value is zero, `ATK$DDP_WRITE` will fill in the local node number and network number of the port on which the datagram is to be transmitted. Once the internet address block is complete, `ATK$DDP_WRITE` will copy it to the DDP header of the datagram buffer. If the value of `src-netaddr` is zero, `ATK$DDP_WRITE` will not alter the source address in the DDP buffer (this feature is intended to support internetwork router processes that transmit datagrams originating on remote nodes).

The destination information in the `dst-netaddr` argument is used to determine the immediate LAP address to which to send the datagram. On a multiport node (that is, a bridge), the decision requires a full routing table. On a single-port node, the default routing stub is enough. The call to `ATK$INIT_ATALK` establishes the full/stub routing environment.

If the destination and source net numbers are equal to each other and to the net number of the port on which the datagram is to be transmitted, the library converts the datagram into the "short" form. *This conversion is destructive; you cannot depend on the contents of a datagram buffer following its use by `ATK$DDP_WRITE`.*

The datagram is returned to the silo list when transmission actually completes if the `completion-routine` argument is `NULL` and the `async-flag` argument is nonzero (signifying asynchronous transmission). Otherwise, the client is responsible for returning the DDP datagram to the silo list when the transmission completes. Handling the allocation and deallocation of DDP datagrams in this manner allows the client to reuse datagrams without having to deallocate and reallocate them between each transmission.

For more information, consult *Inside AppleTalk*.

Conditions

<code>SS\$_NORMAL</code>	Routine successfully completed.
<code>ATK__SKTNOTOPN</code>	Source socket number for transmit is not open.
<code>ATK__BUFTOOBIG</code>	Specified datagram length is too big. DDP allows for a maximum datagram length of 586 bytes.
<code>ATK__NETNOTREA</code>	The specified destination network number is not reachable. Only returned if library was initialized as a full router.

Any condition values returned by the following AppleTalk Protocol Library routines:

`ATK$LAP_XMIT`

ATK\$ENTITY_LEN – Return length of NBP entity

This routine returns the length, in bytes, of an NBP entity.

Format `ATK$ENTITY_LEN    entity`

Returns type: word (unsigned)
 access: write only
 mechanism: by value

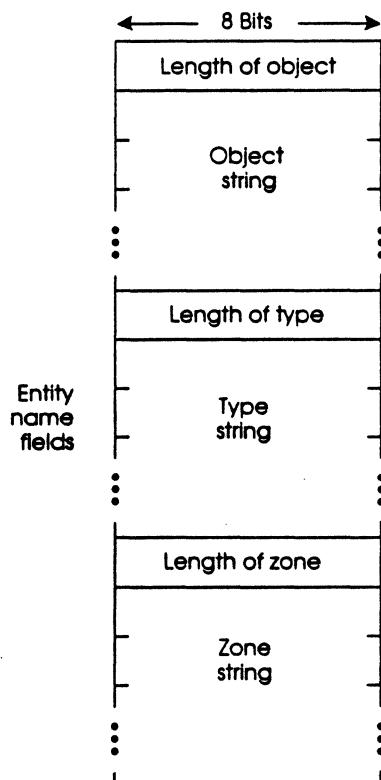
The total length, in bytes, of the NBP entity including the count bytes.

Arguments `entity`

type: unspecified
access: read only
mechanism: by reference

The entity to return the length of. The `entity` argument contains the address of the entity buffer.

Description This routine calculates the length of an NBP entity. The NBP entity must have the following format:



ATK\$EP_REQ – Send an EP request

This routine sends an Echo Protocol (EP) request to the echoer socket in the specified destination node.

Format

ATK\$EP_REQ	net-address
	, request-buffer
	, request-buf-size
	, elapsed-time
	[, retry-count
	[, retry-interval]]

Returns

type:	longword (unsigned)
access:	write only
mechanism:	by value

One of the values listed below under "Conditions."

Arguments

net-address
type: unspecified
access: write only
mechanism: by reference

The 4-byte internet address of the echoer process. The **net-address** argument is the address of the internet address block (see "Description" below). The first 2 bytes contain the 16-bit network address. The first byte is the least significant byte, followed by the most significant byte. The third byte contains the node number, and the fourth contains the socket number. The socket number in this argument is not used, as all EP requests are sent to the statically assigned, well-known echoer socket number 4.

request-buffer

type: unspecified
access: read only
mechanism: by reference

The buffer containing the request data to send with the EP request. The **request-buffer** argument is the address of this buffer.

request-buf-size

type: word (unsigned)
access: read only
mechanism: by value

The size in bytes of the request data buffer. The **request-buf-size** argument contains the value of the size of the request buffer. EP allows for a maximum request buffer size of 585 bytes.

elapsed-time

type: quadword (unsigned)
access: write only
mechanism: by reference

The elapsed time, in VMS system quadword format, for the EP request/reply to complete. The **elapsed-time** argument is the address of a quadword to receive the elapsed time.

retry-count

type: word (unsigned)
access: read only
mechanism: by value

The number of times to retry the request. The **retry-count** argument contains the value of the retry count. If **retry-count** is not specified, a default value of 8 is used. A **retry-count** value of zero means the request will be issued only once, and the process will wait for the number of seconds specified by the **retry-interval** argument.

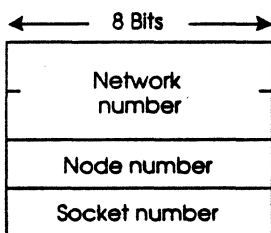
retry-interval

type: word (unsigned)
access: read only
mechanism: by value

The interval, in seconds, between request retries. The **retry-interval** argument contains the value of the retry interval. If **retry-interval** is not specified or is zero, a default value of 2 is used.

Description

This routine provides a client process with the Echo Protocol service. A client process wishing to use this service must first determine the internet address of the node it wishes to obtain the echo from (it will probably use NBP for this). The address of the internet address block is one of the parameters specified in the argument list. The internet address block contains an internet address as follows:



The socket number in the internet address block is not used by this routine, as the request must be sent to the statically assigned, well-known echoer socket number 4.

For more information, consult *Inside AppleTalk*.

Conditions	SS\$_NORMAL	Routine successfully completed.
	ATK__RTYCNTEXC	No echo reply received before the retry count expired.
	ATK__REQTOOBIG	Request buffer is too big.

Any condition values returned by the following AppleTalk Protocol Library routines:

ATK\$GET_DATAGRAM

ATK\$DDP_OPN_SKT

ATK\$DDP_WRITE

ATK\$TMRINS

Any condition values returned by the following VMS Run-Time Library routines:

LIB\$INIT_TIMER

LIB\$STAT_TIMER

ATK\$GET_DATAGRAM - Get a datagram buffer

This routine allocates a datagram buffer from the internal buffer pool, removing it from the silo.

Format ATK\$GET_DATAGRAM datagram

Returns type: longword (unsigned)
 access: write only
 mechanism: by value

One of the values listed below under "Conditions."

Arguments **datagram**

 type: unspecified
 access: write only
 mechanism: by reference

The datagram buffer allocated from the internal buffer pool. The **datagram** argument is the address of the location at which to store the datagram buffer pointer.

Description This routine retrieves a buffer from the "front" of the silo list. It is used to get a buffer for use in a new operation. The returned address points to the datagram portion of the buffer. This routine is recommended for use in user programs.

Conditions	SS\$_NORMAL	Routine successfully completed.
	ATK__ONEENTQUE	Routine completed successfully; however, only one entry is remaining.
	ATK__QUEWASEMP	There are no buffers available for allocation.

ATK\$GET_PHYS_ADDR – Get port physical address

This routine returns the physical address by which the specified port is known on its LAP.

```
Format      ATK$GET_PHYS_ADDR      port-id
          , phys-addr
```

Returns	type:	unspecified
	access:	write only
	mechanism:	by reference

One of the values listed below under "Conditions."

Arguments	port-id
	type: longword (unsigned)
	access: read only
	mechanism: by value

The identifier of the port for which to get the physical address. The `port-id` argument contains the value of the port identifier.

phys-addr

```

type:      unspecified
access:    modify
mechanism: by descriptor

```

The returned physical destination address of the given port's LAP. The `phys-addr` parameter is the address of a descriptor that points to the returned physical address. The physical address has a length in bytes appropriate to the particular port's LAP.

Conditions SS\$ NORMAL Routine successfully completed.

Any condition values returned by the following VMS Run-Time Library routines:

LIB\$SCOPY R DX

ATK\$INIT_ATALK – Initialize AppleTalk

A VMS process *must* call this routine prior to using any other AppleTalk for VMS Protocol Library procedure. It establishes the process buffer pool, starts the library's internal timer process scheduler, starts the echo and names information processes, and optionally starts the routing and broadcast name responder processes as required.

Format

```
ATK$INIT_ATALK  buffer-alloc
                  [, router-process
                  [, nbp-brq-proc]]
```

Returns

type: longword (unsigned)
access: write only
mechanism: by value

One of the values listed below under "Conditions."

Arguments

buffer-alloc

type: longword (unsigned)
access: read only
mechanism: by value

Number of datagram buffers to allocate to the buffer pool. The `buffer-alloc` argument contains the number of buffers to allocate.

router-process

type: procedure entry mask
access: call without stack unwinding
mechanism: by reference

The entry mask of the procedure to be called to handle the routing of datagrams. The `router-process` argument is the address of the procedure mask. If supplied, this argument causes the library to be initialized in the configuration for full routing service.

nbp-brq-proc

type: procedure entry mask
access: call without stack unwinding
mechanism: by reference

The entry mask of the procedure to be called to handle incoming NBP broadcast lookup requests. The `nbp-brq-proc` argument is the address of the procedure mask. Required if `router-process` is specified.

Conditions SSS_NORMAL Routine successfully completed.

Any condition values returned by the following AppleTalk Protocol Library routines:

ATK\$DDP_OPN_SKT

ATK\$TMRINS

Any condition values returned by the following VMS Run-Time Library routines:

LIB\$GET_VM

LIB\$GET_EF

ATK\$INIT_PORT – Load and initialize a port driver

This procedure must be called prior to opening a port with `ATK$OPEN_PORT`. `ATK$INIT_PORT` loads the port driver into the process's address space, and calls the driver at its initialization entry point. This is a one-time initialization, performed only at load time.

```
Format      ATK$INIT_PORT      portdriver
                                ,portinfo
```

Returns	type:	longword (unsigned)
	access:	write only
	mechanism:	by value

One of the values listed below under "Conditions."

Arguments **portdriver**

```
type:      character string
access:    read only
mechanism: by descriptor
```

The name of the port driver to open (see Appendix A for information on the individual port drivers). The `portdriver` argument contains the address of a descriptor pointing to this string.

portinfo

```

type:      character string
access:    read only
mechanism: by descriptor

```

The port-specific information string to be parsed to the port driver open routine. The `portinfo` argument contains the address of a descriptor pointing to this string. A zero may be specified if the port information string for the port driver is optional (refer to Appendix A for port driver specifications).

Description	This procedure loads a port driver into process memory. The procedure then calls the driver at its initialization entry point.
--------------------	--

Conditions \$\$\$ NORMAL Routine successfully completed.

ATK MAJIDBAD Major ID mismatch.

ATK MINIDBAD Minor ID mismatch.

Any condition values returned by the following VMS Run-Time Library routines:

LIB\$FIND IMAGE SYMBOL

LIB\$SCOPY DXDX

This routine inserts an entry in a queue. **ATK\$INSQUE** makes the **VAX/VMS INSQUE** instruction available as a callable procedure.

Arguments **entry**

Entry to be inserted by `ATK$INSQUE`. The `entry` argument contains the address of this signed quadword-aligned array, which must be at least 8 bytes long. Bytes following the first 8 bytes can be used for any purpose by the calling program.

type:	quadword integer (signed)
access:	modify
mechanism:	by reference, array reference

The queue entry after which entry is to be inserted. The predecessor argument contains the address of this signed quadword-aligned queue entry.

ATK\$LAP_XMIT - Transmit a datagram

This routine transmits a datagram buffer on the specified port.

Format

```
ATK$LAP_XMIT    buffer
                  ,buffer-length
                  ,port-id
                  ,phys-addr
                  ,async-flag
                  ,completion-routine
                  ,completion-parameter
```

Returns

type: longword (unsigned)
access: write only
mechanism: by value

One of the values listed below under "Conditions."

Arguments

buffer

type: unspecified
access: read only
mechanism: by reference

The buffer containing the LAP-level packet to be queued for transmission. The **buffer** argument contains the address of the buffer. The buffer must have been allocated from the silo buffer (see ATK\$GET_DATAGRAM).

buffer-length

type: longword (unsigned)
access: read only
mechanism: by value

The size, in bytes (up to 603), of the buffer. The **buffer-length** argument contains the value of the buffer size.

port-id

type: longword (unsigned)
access: read only
mechanism: by value

The identifier of the port on which the datagram is to be transmitted. The **port-id** argument contains the value of the port identifier.

phys-addr

type: unspecified
access: read only
mechanism: by reference

The physical destination address to use on the given port's LAP. The **phys-addr** parameter is the address of an array of bytes whose length is appropriate to the particular port's LAP.

async-flag

type: byte (unsigned)
access: read only
mechanism: by value

A flag value indicating whether or not the transaction is to be issued asynchronously; that is, whether the **ATKSLAP_XMIT** procedure returns control to the caller as soon as the transaction request has been issued. The **async-flag** argument contains the value of the flag. Any nonzero value indicates asynchronous mode.

completion-routine

type: procedure entry mask
access: call without stack unwinding
mechanism: by reference

The entry mask of a procedure that is called when the transmit completes. The **completion-routine** argument contains the address of this procedure entry mask.

This routine is called with the following format:

Format	completion-routine	completion-parameter ,completion-status
---------------	---------------------------	--

Arguments **completion-parameter**

type: longword
access: read only
mechanism: by value

completion-status

type: longword
access: read only
mechanism: by value

The library calls the completion routine with the **completion-parameter** and the **completion-status** of the transmit. The completion routine must do a normal return when it completes. A zero may be specified if no completion routine is desired.

completion-parameter

type: longword
access: read only
mechanism: by value

A caller-supplied value that is passed as an argument to the completion routine. The completion-parameter argument is a longword containing the caller-supplied value.

Description

This procedure transmits a LAP packet on the network defined by the specified port ID. The datagram buffer containing the transmitted datagram is returned to the silo list when transmission actually completes if the completion-routine argument is NULL and the async-flag argument is nonzero (signifying asynchronous transmission). Otherwise, the client is responsible for returning the datagram buffer to the silo list when the transmission completes. Handling the allocation and deallocation of datagram buffers in this manner allows the client to reuse datagram buffers without having to deallocate and reallocate them between each transmission.

Conditions

SS\$NORMAL Routine successfully completed.

ATK__PRTISCLS The port associated with the specified port ID has been closed.

Any condition values returned by the following VMS system services:

SYSSQIO

SYSSQIOW

ATK\$MAKE_ENTITY - Construct an NBP entity

This routine constructs an NBP entity name from object, type, and zone string descriptors.

Format

```
ATK$MAKE_ENTITY  entity
                  ,entitylen
                  ,object-str
                  ,type-str
                  ,zone-str
```

Returns

type: longword (unsigned)
access: write only
mechanism: by value

One of the values listed below under "Conditions."

Arguments

entity

type: unspecified
access: modify
mechanism: by reference

The buffer that receives the constructed NBP entity name. The entity argument contains the address of the buffer.

entitylen

type: word (unsigned)
access: write only
mechanism: by reference

The size, in bytes, of the constructed NBP entity name. The entitylen argument contains the address of a word to receive the size.

object-str

type: character string
access: read only
mechanism: by descriptor

The object field of the NBP entity name to be constructed. The object-str argument contains the address of a descriptor pointing to this string. The metacharacter "=" is a wildcard, meaning "all objects."

type-str

type: character string
access: read only
mechanism: by descriptor

The type field of the NBP entity name to be constructed. The type-str argument contains the address of a descriptor pointing to this string. The metacharacter "=" is a wildcard, meaning "all types."

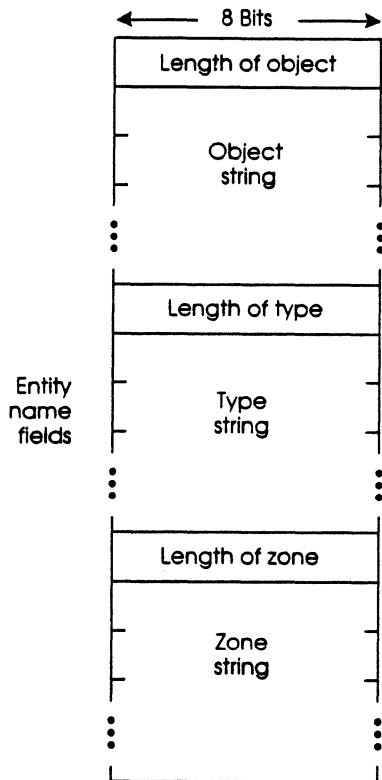
zone-str

type: character string
access: read only
mechanism: by descriptor

The zone field of the NBP entity name to be constructed. The zone-str argument contains the address of a descriptor pointing to this string. The metacharacter "*" means "this zone."

Description

This routine constructs an NBP entity from the supplied object, type, and zone descriptors. The constructed NBP entity has the following format:



Conditions

SS\$_NORMAL

Routine successfully completed.

Any condition values returned by the following VMS Run-Time Library routines:

LIB\$SCOPY_DXDX

ATK\$NBP_CONFIRM – Confirm name via NBP

This routine checks to see if an entity name is still registered at the specified internet address. If the name is still registered on the node but at a different socket number, the socket number in the entity tuple is updated.

Format

```
ATK$NBP_CONFIRM  entity-tuple
                  ,port-id
                  [,trans-count
                  [,trans-interval]]
```

Returns

type: longword (unsigned)
access: write only
mechanism: by value

One of the values listed below under "Conditions."

Arguments

entity-tuple

type: unspecified
access: modify
mechanism: by reference

The buffer that contains the entity tuple. The entity-tuple argument contains the address of the entity tuple. No metacharacters are allowed in the entity name strings.

port-id

type: longword (unsigned)
access: read only
mechanism: by value

The identifier of the port on which the confirm request is to be transmitted. The port-id argument contains the value of the port identifier.

trans-count

type: word (unsigned)
access: read only
mechanism: by value

The number of times to transmit the confirmation request. The trans-count argument contains the value of the transmission count. If trans-count is not specified or is zero, a default value of 8 is used.

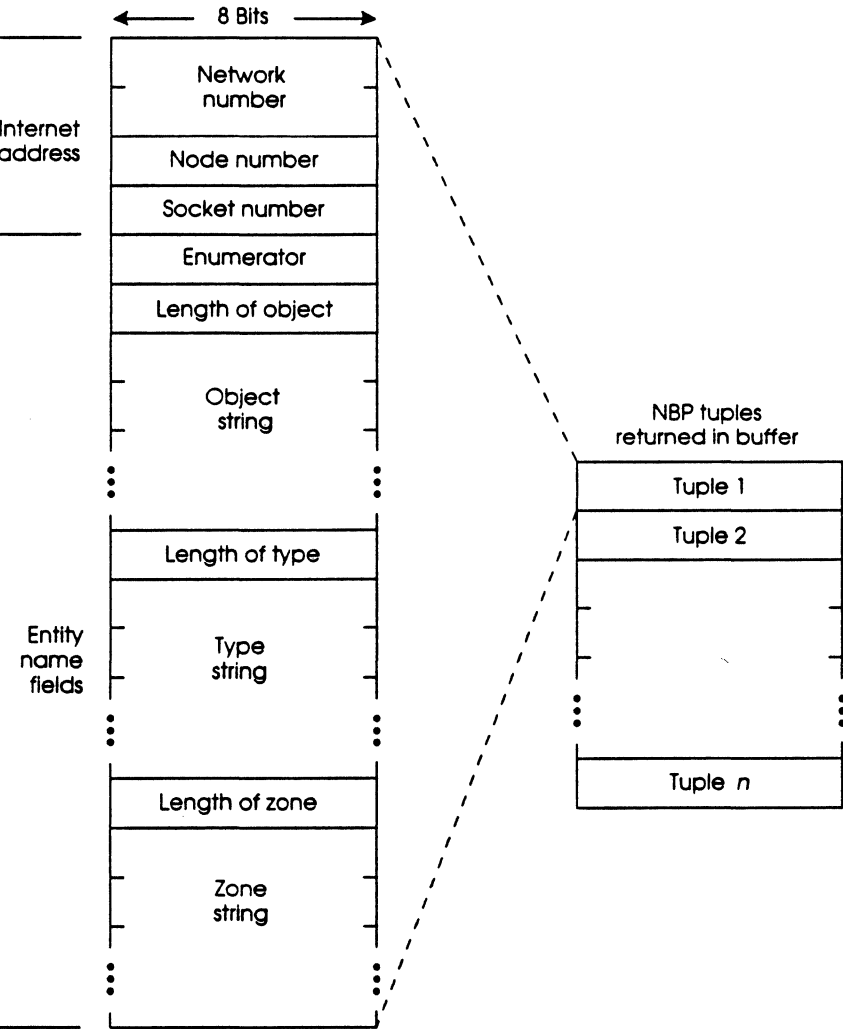
trans-interval

type: word (unsigned)
access: read only
mechanism: by value

The interval, in seconds, between confirmation requests. The trans-interval argument contains the value of the transmission interval. If trans-interval is not specified or is zero, a default value of 1 second is used.

Description

This routine sends an NBP confirmation request to the specified net/node to confirm the registration of the specified name. The address and name are supplied as an entity tuple, as shown below. The socket number may be altered if the named object has changed socket numbers.



Conditions	SS\$ _NORMAL	Routine successfully completed.
	ATK __CNFNEWSKT	Success, confirmed with new socket number.
	ATK __NAMNOTCNF	Name was not confirmed.
	ATK __INVENTNAM	Either the object, type, or zone field contained illegal characters or was too long.

Any condition values returned by the following AppleTalk Protocol Library routines:

ATK\$GET_DATAGRAM

ATK\$DDP_OPN_SKT

ATK\$DDP_WRITE

ATK\$TMRINS

ATK\$NBP_LOOKUP - Look up name(s) via NBP

This routine returns a list of registered name-address pairs via an NBP lookup on the specified port.

Format

```
ATK$NBP_LOOKUP  object-str
                  ,type-str
                  ,zone-str
                  ,port-id
                  ,return-buffer
                  ,return-size
                  ,max-to-get
                  ,number-gotten
                  [,trans-count
                  [,trans-interval]]
```

Returns

type: longword (unsigned)
access: write only
mechanism: by value

One of the values listed below under "Conditions."

Arguments **object-str**

type: character string
access: read only
mechanism: by descriptor

The object field of the name to be looked up. The `object-str` argument contains the address of a descriptor pointing to this string. The metacharacter "=" is a wildcard, meaning "all objects."

type-str

type: character string
access: read only
mechanism: by descriptor

The type field of the name to be looked up. The `type-str` argument contains the address of a descriptor pointing to this string. The metacharacter "=" is a wildcard, meaning "all types."

zone-str

type: character string
access: read only
mechanism: by descriptor

The zone field of the name to be looked up. The `zone-str` argument contains the address of a descriptor pointing to this string. The metacharacter "*" means "this zone."

port-id

type: longword (unsigned)
access: read only
mechanism: by value

The identifier of the port on which the lookup request is to be transmitted. The **port-id** argument contains the value of the port identifier.

return-buffer

type: unspecified
access: modify
mechanism: by reference

The buffer that receives concatenated entity tuples. The **return-buffer** argument contains the address of the buffer.

return-size

type: word (unsigned)
access: read only
mechanism: by value

The size, in bytes, of the buffer that receives concatenated entity tuples. The **return-size** argument contains the value of the buffer size in bytes.

max-to-get

type: word (unsigned)
access: read only
mechanism: by value

The maximum number of entity tuples to accept. The **max-to-get** argument contains the value of the maximum number of unique entity tuples to accept from the lookup broadcast. If **max-to-get** tuples are received before **trans-count** broadcasts have been done, the lookup terminates.

number-gotten

type: word (unsigned)
access: write only
mechanism: by reference

The total number of unique entity tuples actually received as a result of the lookup request. The **number-gotten** argument contains the address of a word that receives this count.

trans-count

type: word (unsigned)
access: read only
mechanism: by value

The number of times to broadcast a request for the name. The **trans-count** argument contains the value of the broadcast count. If **trans-count** is not specified or is zero, a default value of 8 is used.

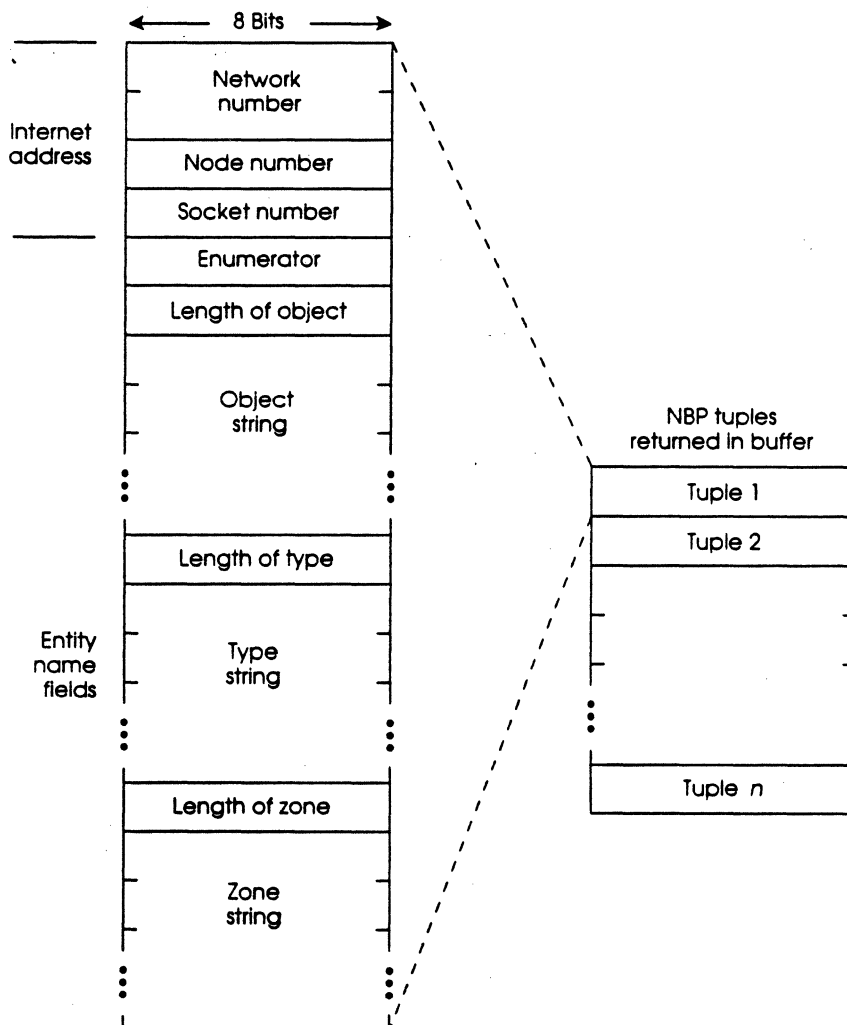
trans-interval

type: word (unsigned)
access: read only
mechanism: by value

The interval, in seconds, between broadcast lookups for the name. The `trans-interval` argument contains the value of the transmission interval. If `trans-interval` is not specified or is zero, a default value of 1 second is used.

Description

This routine fills in a buffer with the addresses of all entities with a specified name. The specified name may contain the "=" wildcard metacharacter in either the object or the type field or both. The returned addresses are paired with their corresponding complete NBP names in a buffer and concatenated, with duplicates filtered out. Each name-address pair is placed into the return buffer in the form of an NBP entity tuple, as shown below:



Note that a lookup may return multiple matching names when the object and/or type field contains the "=" wildcard metacharacter. Therefore, the caller must allocate a buffer to receive the incoming entity tuples, and pass the address and length of this buffer to `ATK$NBP_LOOKUP`.

In addition, the caller may specify a tuple maximum; the lookup process terminates when this count is reached, otherwise it completes when the transmit count is exhausted. In any case, the actual number of tuples received is returned to the caller.

Conditions

SS\$ _NORMAL	Routine successfully completed.
ATK__BUFOVRFLW	The return buffer was not large enough to receive all of the matching tuples.
ATK__NOBRIDGE	An attempt was made to perform a lookup in another zone and no bridge could be found to service the request.
ATK__INVENTNAM	Either the object, type, or zone field contained illegal characters or was too long.

Any condition values returned by the following AppleTalk Protocol Library routines:

ATK\$MAKE_ENTITY
ATK\$GET_DATAGRAM
ATK\$DDP_OPN_SKT
ATK\$DDP_WRITE
ATK\$TMRINS

This routine adds the specified name-socket pair to the list of registered names on the specified port.

Returns type: longword (unsigned)
 access: write only
 mechanism: by value

One of the values listed below under "Conditions."

The identifier of the port on whose names table the name is to be registered. The `port-id` argument contains the value of the port identifier.

trans-count

type: word (unsigned)
access: read only
mechanism: by value

The number of times to broadcast a request for the name to make sure it isn't already in use. The **trans-count** argument contains the value of the broadcast count. If **trans-count** is not specified or is zero, a default value of 8 is used.

trans-interval

type: word (unsigned)
access: read only
mechanism: by value

The interval, in seconds, between broadcast lookups for the name to make sure it isn't already in use. The **trans-interval** argument contains the value of the transmission interval. If **trans-interval** is not specified or is zero, a default value of 1 second is used.

Description

This routine registers an NBP name on the specified port. Only the object and type fields of the name are needed. The zone field is set to "" (thisZone) automatically. The corresponding internet address is constructed from the supplied socket number and the port ID, which in turn determines the node and network number of the entity.

Note that a name is known in the context of a port. A particular service may have a socket open on a node with multiple active ports. The node is known by separate internet addresses on each port. The service must register its name on each port separately, since each port has its own internet address.

Conditions

SS\$_NORMAL	Routine successfully completed.
ATK__NAMINUSE	The requested name is in use, either locally or on another node in the zone.
ATK__INVENTNAM	Either the object or type field contained illegal characters, such as NBP metacharacters, or was too long.

Any condition values returned by the following AppleTalk Protocol Library routines:

ATK\$MAKE_ENTITY

ATK\$GET_DATAGRAM

ATK\$DDP_OPN_SKT

ATK\$DDP_WRITE

ATK\$TMRINS

Any condition values returned by the following VMS Run-Time Library routines:

LIB\$GET_VM

This routine removes the specified entity name from the list of registered names on the specified port.

Returns	type:	longword (unsigned)
	access:	write only
	mechanism:	by value

Arguments `object-str`

The object field of the name to be removed. The `object-str` argument contains the address of a descriptor pointing to this string. No metacharacters are allowed.

```

type:      character string
access:    read only
mechanism: by descriptor

```

The type field of the name to be removed. The type-str argument contains the address of a descriptor pointing to this string. No metacharacters are allowed.

```

type:      longword (unsigned)
access:    read only
mechanism: by value

```

The identifier of the port from whose names table the name is to be removed. The `port-id` argument contains the value of the port identifier.

ATK\$NBP_REMOVE

Conditions**SS\$_NORMAL**

Routine successfully completed.

ATK__NAMNOTREG

The name is not registered on the specified port.

ATK__INVENTNAM

Either the object or type field contained illegal characters or was too long.

Any condition values returned by the following AppleTalk Protocol Library routines:

ATK\$MAKE_ENTITY

ATK\$NET_NODE - Get AppleTalk address

This routine returns the AppleTalk network and node number by which the specified port is known on its LAP.

Format ATK\$NET_NODE port-id
 , network
 , node

Returns type: longword (unsigned)
 access: write only
 mechanism: by value

One of the values listed below under "Conditions."

Arguments **port-id**
 type: longword (unsigned)
 access: read only
 mechanism: by value

The identifier of the port from which to obtain the AppleTalk address. The **port-id** argument contains the value of the port identifier.

network
type: word (unsigned)
access: write only
mechanism: by reference

The AppleTalk network number of the specified port. The **network** argument contains the address of an unsigned word that receives the AppleTalk network number.

node
type: byte (unsigned)
access: write only
mechanism: by reference

The AppleTalk node number of the specified port. The **node** argument contains the address of an unsigned byte that receives the AppleTalk node number.

Conditions **SS\$_NORMAL** Routine successfully completed.

ATK\$OPEN_PORT - Open port

This procedure starts port driver operations, performs device-specific startup, and initializes the port descriptor block by filling in port descriptor fields. Refer to Appendix A for information on the individual port drivers.

Format

```
ATK$OPEN_PORT    port-id
                  ,portdriver
                  ,portinfo
                  ,portname
```

Returns

type: longword (unsigned)
access: write only
mechanism: by value

One of the values listed below under "Conditions."

Arguments

port-id

type: longword (unsigned)
access: write only
mechanism: by reference

The numeric identifier to be used with this port. The `port-id` argument is the address of an unsigned longword that receives the port ID.

portdriver

type: character string
access: read only
mechanism: by descriptor

The name of the port driver to open (see Appendix A for information on the individual port drivers). The `portdriver` argument is the address of a descriptor pointing to this string.

portinfo

type: character string
access: read only
mechanism: by descriptor

The port-specific information string to be passed to the port driver open routine. The `portinfo` argument is the address of a descriptor pointing to this string. A zero may be specified if the port information string for the port driver is optional (refer to Appendix A).

portname

type: character string
access: read only
mechanism: by descriptor

The name to be assigned to the port. The `portname` argument is the address of a descriptor pointing to this string.

Description This procedure establishes AppleTalk communication between the calling process and a port. It initializes the port-specific data structures. If the opened port is an AppleTalk-speaking port, this routine issues an `ATK$RTMP_REQ` in an attempt to discover the port's network number and the node ID of a bridge on the network.

Conditions `SS$_NORMAL` Routine successfully completed.

`ATK__NAMINUSE` The port name is in use

Any condition values returned by the specific port's open routine.

Any condition values returned by the following VMS Run-Time Library routines:

`LIB$FIND_IMAGE_SYMBOL`

`LIB$GET_VM`

`LIB$SCOPY_DXDX`

ATK\$PAP_CLOSE – Close a PAP connection

A PAP client in a workstation calls this routine to close a connection specified by a reference number. This routine completes any and all pending ATK\$PAP_READ or ATK\$PAP_WRITE calls for the connection with a "connection closed" status. It also calls the close routine specified in the ATK\$PAP_OPEN call with this status.

Format ATK\$PAP_CLOSE refnum

Returns type: longword (unsigned)
 access: write only
 mechanism: by value

One of the values listed below under "Conditions."

Arguments refnum

 type: longword (unsigned)
 access: read only
 mechanism: by value

The connection reference number of an opened PAP connection that the client wishes to close. The refnum argument contains the value of the connection reference number.

Conditions SS\$_NORMAL Routine successfully completed.

 ATK__INVREFNUM Invalid connection reference number.

ATK\$PAP_OPEN – Open a PAP connection

A PAP client in a workstation calls this routine to open a connection to a specified PAP server.

Format

ATK\$PAP_OPEN	port-id , refnum , server-obj-str , server-type-str , server-zone-str , server-adr , flowquantum , server-flowquantum , status-buf , status-routine , status-param , completion-routine , completion-param , close-routine
---------------	---

Returns

type:	longword (unsigned)
access:	write only
mechanism:	by value

One of the values listed below under "Conditions." Note that unless the description states otherwise, all condition values except `SS$_NORMAL` are only returned from the routine or passed as a status value to the completion routine.

Arguments

port-id	
type:	longword (unsigned)
access:	read only
mechanism:	by value

The identifier of the port on which the connection request is to be issued. The `port-id` argument contains the value of the port identifier.

refnum	
type:	longword (unsigned)
access:	write only
mechanism:	by reference

The connection reference number of the opened PAP connection. The `refnum` argument contains the address of a longword that receives the connection reference number.

server-obj-str

type: character string
access: read only
mechanism: by descriptor

The object field of the PAP server name to which the connection is to be opened. The **server-obj-str** argument contains the address of a descriptor pointing to this string. The metacharacter "=" is a wildcard, meaning the first server in the **server-zone-str** zone with **server-type-str** type to respond to the connection request.

server-type-str

type: character string
access: read only
mechanism: by descriptor

The type field of the PAP server name to which the connection is to be opened. The **server-type-str** argument contains the address of a descriptor pointing to this string. The metacharacter "=" is a wildcard, meaning the first server in the **server-zone-str** zone of **server-obj-str** object to respond to the connection request.

server-zone-str

type: character string
access: read only
mechanism: by descriptor

The zone field of the PAP server name to which the connection is to be opened. The **server-zone-str** argument contains the address of a descriptor pointing to this string. The metacharacter "*" means "this zone."

server-adr

type: unspecified
access: write only
mechanism: by reference

The 4-byte internet address of the PAP server to which the connection has been requested. The **server-adr** argument is the address of an internet address block (see "Description" below) that receives the PAP server's internet address. The high-order 16 bits contain the network number, the next least significant byte is the node number, and the least significant byte is the socket number of the PAP server's service listening socket. A zero may be specified if you are not interested in the server's internet address.

flowquantum

type: byte (unsigned)
access: read only
mechanism: by value

The maximum number of 512-byte PAP buffers available in the workstation for receiving data from the PAP server. The **flowquantum** argument is the value of this number. For example, if **flowquantum** equals N, then the number of bytes available in the client is $512 * N$ bytes. The Apple LaserWriter uses 8.

type:	byte (unsigned)
access:	write only
mechanism:	by reference

status-buf

The buffer in which the PAP server's status is returned to the caller during the opening process. The `status-buf` argument is the address of a character string descriptor pointing to this buffer. The library calls the client's status routine whenever this buffer changes. The buffer must be at least 255 bytes. The library fills in this argument's descriptor length field with the actual length of the status string.

type:	procedure entry mask
access:	call without stack unwinding
mechanism:	by reference

This routine is called with the following format:

Arguments **status-param**

```

type:      longword
access:    read only
mechanism: by value

```

status-code

```

type:      longword
access:    read only
mechanism: by value

```

ATKSPAP OPEN

status-param

type: longword (unsigned)
access: read only
mechanism: by value

A caller-supplied value that is passed as an argument to the status routine. The **status-param** argument is a longword containing the caller-supplied value.

completion-routine

type: procedure entry mask
access: call without stack unwinding
mechanism: by reference

The entry mask of a procedure that is called when the open connection request completes. The **completion-routine** argument contains the address of this procedure entry mask.

This routine is called with the following format:

Format	completion-routine	completion-parameter , completion-status
---------------	---------------------------	---

Arguments	completion-parameter
------------------	-----------------------------

type:	longword
access:	read only
mechanism:	by value

	completion-status
--	--------------------------

type:	longword
access:	read only
mechanism:	by value

The library calls the completion routine with the **completion-parameter** and the **completion-status** of the open request. The completion routine must do a normal return when it completes. A zero may be specified if no completion routine is desired.

completion-param

type: longword
access: read only
mechanism: by value

A caller-supplied value that is passed as an argument to the completion routine. The **completion-param** argument is a longword containing the caller-supplied value.

close-routine

type: procedure entry mask
access: call without stack unwinding
mechanism: by reference

The entry mask of a procedure that is called when an opened connection is closed. The **close-routine** argument contains the address of this procedure entry mask.

This routine is called with the following format:

Format	<code>close-routine</code>	<code>refnum</code> <code>, status-code</code>
---------------	----------------------------	---

Arguments	<code>refnum</code>
	type: longword
	access: read only
	mechanism: by value
	<code>status-code</code>
	type: longword
	access: read only
	mechanism: by value

The library calls the close routine with the connection `refnum` and a `status-code` indicating the reason that the library closed the connection. The close routine must do a normal return when it completes. A zero may be specified if no close routine is desired.

Description

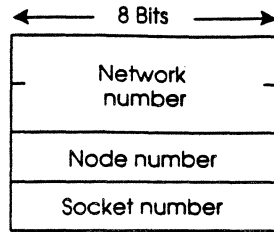
A PAP client in a workstation calls this routine to open a connection to a specified PAP server. `ATK$NBP_LOOKUP` is used to find the internet address of the specified PAP server. The NBP call is issued with a transmission count and interval of 5.

This routine is executed asynchronously, which means control is returned to the caller as soon as the open request has been sent to the PAP server. The library informs the client when the connection is established or aborted through a client-supplied completion routine.

In addition, the library informs the client of the connection status through a client-supplied status buffer and status routine. Whenever the contents of the status buffer change, the library calls the client's status routine. The client may wish to display this string in an appropriate fashion to provide feedback to the user. The client's status routine may terminate an open request by returning a status code of other than `SS$_NORMAL`.

Either end of a PAP connection may close the connection for various reasons, some of which are outside the control of the client in the workstation. For this reason, whenever the library closes a connection, it calls the client-supplied close routine. The library calls the close routine after it cancels any and all pending PAP I/O for the connection.

Once a connection is established, a PAP client in the workstation may wish to send periodic status requests to the PAP server using the `ATK$PAP_STATUS` call. `ATK$PAP_STATUS` requires the caller to supply the entity name of the PAP server or the internet address of the PAP server's service listening socket. Since the `ATK$PAP_OPEN` call must obtain the internet address of the PAP server to issue the open request, it returns, as a convenience to the caller, the internet address of the PAP server. The address is returned in an internet address block. The internet address block has the following format:



For more information, consult *Inside AppleTalk*.

Conditions	SS\$_NORMAL	Routine successfully completed.
	ATK__PRTBSY	PAP server client is busy (status routine only).
	ATK__RTYCNTEXC	Retry count exceeded while attempting to look for the specified server (status routine only).
	ATK__TOOMNYCON	Too many open connections requested. PAP allows for a maximum of 255 connections.
	ATK__OPNABO	Open request was aborted by client returning from status routine with other than SS\$_NORMAL.
	ATK__CONCLS	Connection closed by a client call to ATK\$PAP_CLOSE (close routine only).
	ATK__CLSCONREQ	Connection closed at the request of the PAP server (close routine only).
	ATK__CONTMREXP	Connection closed as a result of the connection timer expiring (close routine only).

Any condition values returned by the following AppleTalk Protocol Library routines:

ATK\$NBP_LOOKUP

ATK\$ATP_OPN_SKT

ATK\$ATP_SND_REQ

ATK\$ATP_GET_REQ

ATK\$TMRINS

Any condition values returned by the following VMS Run-Time Library routines:

LIB\$GET_VM

ATK\$PAP_READ – Read over a PAP connection

A PAP client at either end of a connection calls this routine to read data from the other end over the connection specified by the reference number.

Format

ATK\$PAP_READ	refnum
	, read-buf
	, read-buf-size
	, eof-flag
	, completion-routine
	, completion-param

Returns

type:	longword (unsigned)
access:	write only
mechanism:	by value

One of the values listed below under "Conditions."

Arguments

refnum	
type:	longword (unsigned)
access:	read only
mechanism:	by value

The connection reference number of an opened PAP connection over which data is to be read. The **refnum** argument contains the value of the connection reference number.

read-buf

type:	unspecified
access:	read only
mechanism:	by reference

The buffer into which the received data is to be read. The **read-buf** argument is the address of this buffer. This routine assumes that the size of this buffer is equal to the client's flowquantum value times 512.

read-buf-size

type:	word (unsigned)
access:	read only
mechanism:	by value

The size, in bytes, of the received data. The **read-buf-size** argument contains the address of a word to be filled in with the size of the received data.

eof-flag

type: byte
access: write only
mechanism: by reference

A flag value indicating whether or not the data was received with the PAP end-of-file flag set. The **eof-flag** argument contains the address of a byte to receive the end-of-file indication. Any nonzero value indicates end-of-file.

completion-routine

type: procedure entry mask
access: call without stack unwinding
mechanism: by reference

The entry mask of a procedure that is called when data arrives from the other end. The **completion-routine** argument contains the address of this procedure entry mask. The library calls the client's completion routine with two arguments (by value) in the following order: the **completion-param** and the **completion-status** of the read data request. The completion routine must do a normal return when it completes.

completion-param

type: longword
access: read only
mechanism: by value

A caller-supplied value that is passed as an argument to the completion routine. The **completion-param** argument is a longword containing the caller-supplied value.

Description

A PAP client at either end of a connection calls this routine to read data from the other end over the connection specified by the reference number. The routine is executed asynchronously, which means control is returned to the caller as soon as the request has been queued for sending. The library informs the caller of the request's completion through a client-supplied completion routine.

For more information, consult *Inside AppleTalk*.

Conditions

SS\$ _NORMAL	Routine successfully completed.
ATK__INVREFNUM	Invalid connection reference number.
ATK__EXCFLWQTM	Exceeded flowquantum specified in ATK\$PAP_OPEN .
ATK__CONCLS	Request aborted by caller calling ATK\$PAP_CLOSE .
ATK__CLSCONREQ	Request aborted by the other end requesting that the connection be closed.
ATK__CONTMREXP	Request aborted as a result of the connection timer expiring.

Any condition values returned by the following AppleTalk Protocol Library routines:

ATK\$ATP_SND_REQ

Any condition values returned by the following VMS Run-Time Library routines:

LIB\$GET_VM

A PAP client in a server calls this routine to register a server name on the server's service listening socket.

Returns	type:	longword (unsigned)
	access:	write only
	mechanism:	by value

```

type:      longword (unsigned)
access:    read only
mechanism: by value

```

The type field of the PAP server name to register on the service listening socket. The `server-type-str` argument contains the address of a descriptor pointing to this string.

ATKSPAP_REG_NAME

Conditions

SS\$_NORMAL Routine successfully completed.

ATK__INVREFNUM Invalid connection reference number.

Any condition values returned by the following AppleTalk Protocol Library routines:

ATK\$NBP_REGISTER

ATK\$PAP_REM_NAME - Remove a server name

A PAP client in a server calls this routine to deregister a server name from the server's service listening socket.

Format

```
ATK$PAP_REM_NAME s1-refnum
                  ,server-obj-str
                  ,server-type-str
```

Returns

type: longword (unsigned)
access: write only
mechanism: by value

One of the values listed below under "Conditions."

Arguments

s1-refnum

type: longword (unsigned)
access: read only
mechanism: by value

The service listener reference number assigned by the ATK\$PAP_SLINIT routine. The s1-refnum argument contains the value of the service listener reference number.

server-obj-str

type: character string
access: read only
mechanism: by descriptor

The object field of the PAP server name to deregister from the service listening socket. The server-obj-str argument contains the address of a descriptor pointing to this string.

server-type-str

type: character string
access: read only
mechanism: by descriptor

The type field of the PAP server name to deregister from the service listening socket. The server-type-str argument contains the address of a descriptor pointing to this string.

Conditions

SS\$ _NORMAL Routine successfully completed.

ATK__INVREFNUM Invalid connection reference number.

Any condition values returned by the following AppleTalk Protocol Library routines:

ATK\$NBP_REMOVE

ATK\$PAP_SLINIT – Initialize a PAP server

A PAP client in a server calls this routine to initialize the server's service listening socket and to register the server's name on this socket.

Format

ATK\$PAP_SLINIT	port-id ,sl-refnum ,server-obj-str ,server-type-str ,flowquantum ,status-buf ,status-routine ,nextjob-buf ,nextjob-routine ,startjob-routine ,close-routine
-----------------	---

Returns

type:	longword (unsigned)
access:	write only
mechanism:	by value

One of the values listed below under "Conditions."

Arguments

port-id

type:	longword (unsigned)
access:	read only
mechanism:	by value

The identifier of the port on which the server is to register itself. The `port-id` argument contains the value of the port identifier.

sl-refnum

type:	longword (unsigned)
access:	write only
mechanism:	by reference

The service listener reference number assigned to the initialized service listener. The `sl-refnum` argument contains the address of a longword that receives the service listener reference number.

server-obj-str

type:	character string
access:	read only
mechanism:	by descriptor

The object field of the PAP server name to register on the service listening socket. The `server-obj-str` argument contains the address of a descriptor pointing to this string.

server-type-str

type: character string
access: read only
mechanism: by descriptor

The type field of the PAP server name to register on the service listening socket. The **server-type-str** argument contains the address of a descriptor pointing to this string.

flowquantum

type: byte (unsigned)
access: read only
mechanism: by value

The maximum number of 512-byte PAP buffers available in the PAP server for receiving data from a workstation. The **flowquantum** argument is the value of this number. For example, if **flowquantum** equals N, then the number of bytes available in the server is $512 * N$ bytes. The Apple LaserWriter uses 8.

status-buf

type: character string
access: write only
mechanism: by descriptor

The status buffer that the PAP server fills with the server status information string. The **status-buf** argument is the address of a character string descriptor pointing to this buffer. The library calls the server's status routine whenever the PAP server's status is being requested. This gives the server a chance to update the status buffer. The maximum size of the buffer is 255 bytes. The library uses this character string descriptor's length field as the actual length of the status string returned in response to a status request.

status-routine

type: procedure entry mask
access: call without stack unwinding
mechanism: by reference

The entry mask of a procedure that is called when the status of the PAP server is being requested. The **status-routine** argument contains the address of this procedure entry mask. The library passes (by value) the **sl-refnum** to the client's status routine. This enables the client to identify on which service listening socket the status is being requested.

nextjob-buf

type: character string
access: write only
mechanism: by descriptor

The status buffer that the PAP server fills with the server status information string when a PAP open connection request is received on the server's service listening socket. The nextjob-buf argument is the address of a character string descriptor pointing to this buffer. The library calls the server's nextjob routine whenever a PAP open connection request arrives at the service listening socket. This gives the server a chance to fill in the nextjob-buf with the desired status information string. The library returns the contents of the nextjob-buf to the open connection requestor. The maximum size of the buffer is 255 bytes. The library uses this character string descriptor's length field as the actual length of the status string returned in response to the open connection request.

nextjob-routine

type: procedure entry mask
access: call without stack unwinding
mechanism: by reference

The entry mask of a procedure that is called when a PAP open connection request arrives at the server's service listening socket. The nextjob-routine argument contains the address of the procedure entry mask.

This routine is called with the following format:

Format	nextjob-routine	sl-refnum ,refnum ,flowquantum ,connection-wait-time ,nextjob-status
---------------	-----------------	--

Arguments	sl-refnum
	type: longword
	access: read only
	mechanism: by value
	refnum
	type: longword
	access: read only
	mechanism: by value
	flowquantum
	type: longword
	access: read only
	mechanism: by value
	connection-wait-time
	type: longword
	access: read only
	mechanism: by value

```

nextjob-status
type:          longword
access:        read only
mechanism:     by value

```

The library calls the server's nextjob routine with the sl-refnum, the connection refnum (the reference number that will be assigned to the connection when it is completely established), the workstation's flowquantum, the workstation's open connection-wait-time, and the nextjob-status (the status of the service listening socket's nextjob request). If the library calls the nextjob routine with a status of other than SSS_NORMAL, then an internal fatal error has occurred and the service listening socket will be shut down upon return from the nextjob routine. The client is responsible for handling any recovery from such errors. The client's nextjob routine can reject an open connection request by returning a status code of other than SSS_NORMAL. The library sends an open connection reply to the PAP workstation with a PAP result code of "printer busy" (FFFF hexadecimal) when the nextjob routine rejects the open connection request.

startjob-routine

```

type:          procedure entry mask
access:        call without stack unwinding
mechanism:     by reference

```

The entry mask of a procedure that is called when a PAP connection has been established as a result of an incoming open connection request and a return of SSS_NORMAL from the server's nextjob routine. The startjob-routine argument contains the address of this procedure entry mask.

This routine is called with the following format:

```

Format      startjob-routine      sl-refnum
                                         ,refnum
                                         ,startjob-status

```

```

Arguments  sl-refnum
           type:          longword
           access:        read only
           mechanism:     by value

           refnum
           type:          longword
           access:        read only
           mechanism:     by value

           startjob-status
           type:          longword
           access:        read only
           mechanism:     by value

```



If the server's nextjob routine returns `SS$_NORMAL`, the library proceeds to establish the connection. The library calls the server's startjob routine when it establishes the connection. When the library calls the server's startjob routine, the server may begin to issue PAP I/O over the connection. The server must use the returned connection reference number in any future PAP calls to identify the connection.

Either end of a PAP connection may close a connection for various reasons, some of which are outside the control of the server. For this reason, whenever the library closes a connection, it calls the client-supplied close routine. The library calls the close routine after it cancels any and all pending PAP I/O for the connection.

For more information, consult *Inside AppleTalk*.

Conditions	<code>SS\$_NORMAL</code>	Routine successfully completed.
	<code>ATK__CONCLS</code>	Connection closed by client call to <code>ATK\$PAP_CLOSE</code> (close routine only).
	<code>ATK__CLSCONREQ</code>	Connection closed at the request of the server (close routine only).
	<code>ATK__CONTMREXP</code>	Connection closed as a result of the connection timer expiring (close routine only).

Any condition values returned by the following AppleTalk Protocol Library routines:

`ATK$NBP_REGISTER` (completion only)

`ATK$ATP_OPN_SKT` (completion and startjob routine only)

`ATK$ATP_SND_REQ` (startjob routine only)

`ATK$ATP_GET_REQ` (completion, startjob, and nextjob routines only)

`ATK$TMRINS` (startjob routine only)

Any condition values returned by the following VMS Run-Time Library routines:

`LIB$GET_VM` (completion and startjob routine only)

ATK\$PAP_STATUS – Request PAP server status

A PAP client in a workstation calls this routine to determine the status of the specified PAP server.

Format

ATK\$PAP_STATUS	port-id , server-obj-str , server-type-str , server-zone-str , server-adr , status-buf , completion-routine , completion-param
-----------------	---

Returns

type:	longword (unsigned)
access:	write only
mechanism:	by value

One of the values listed below under "Conditions."

Arguments

port-id	type: longword (unsigned) access: read only mechanism: by value
---------	---

The identifier of the port on which the status request is to be issued. The `port-id` argument contains the value of the port identifier.

server-obj-str

type:	character string
access:	read only
mechanism:	by descriptor

The object field of the PAP server to which the status request is to be sent. The `server-obj-str` argument contains the address of a descriptor pointing to this string. The metacharacter "=" is a wildcard, meaning the first server in the `server-zone-str` zone with `server-type-str` type to respond to the status request. This argument is not used and may be zero, if the `server-adr` internet address block's socket number field is nonzero.

server-type-str

type: character string
access: read only
mechanism: by descriptor

The type field of the PAP server to which the status request is to be sent. The **server-type-str** argument contains the address of a descriptor pointing to this string. The metacharacter "=" is a wildcard, meaning the first server in the **server-zone-str** zone of **server-obj-str** object to respond to the status request. This argument is not used and may be zero, if the **server-adr** internet address block's socket number field is nonzero.

server-zone-str

type: character string
access: read only
mechanism: by descriptor

The zone field of the PAP server to which the status request is to be sent. The **server-zone-str** argument contains the address of a descriptor pointing to this string. The metacharacter "*" means "this zone." This argument is not used and may be zero, if the **server-adr** internet address block's socket number field is nonzero.

server-adr

type: unspecified
access: modify
mechanism: by reference

The 4-byte internet address of the PAP server to which the status request is to be sent. The **server-adr** argument is the address of the PAP server's internet address block (see "Description" below). The first 2 bytes contain the 16-bit network address. The first byte is the least significant byte, followed by the most significant byte. The third byte contains the node number, and the fourth contains the socket number of the PAP server's service listening socket. If the socket number field is nonzero, the internet address block is assumed to have a valid service listening socket internet address. If it is zero, the routine uses the **server-obj-str**, **server-type-str**, and **server-zone-str** arguments to discover the PAP server's service listening socket's internet address. The discovered address is then filled into the address block specified by the **server-adr** argument for use in future **ATKSPAP_STATUS** calls.

status-buf

type: character string
access: write only
mechanism: by descriptor

The buffer in which the server status is returned to the caller. The **status-buf** argument is the address of a character string descriptor pointing to this buffer. This routine fills the actual length of the returned status string into the character string descriptor's length field.

completion-routine

type:	procedure entry mask
access:	call without stack unwinding
mechanism:	by reference

The entry mask of a procedure that is called when the status request completes. The completion-routine argument contains the address of this procedure entry mask.

This routine is called with the following format:

Format	completion-routine	completion-parameter , completion-status
---------------	--------------------	---

```

Arguments      completion-parameter
               type:                longword
               access:              read only
               mechanism:            by value

               completion-status
               type:                longword
               access:              read only
               mechanism:            by value

```

The library calls the completion routine with the completion-parameter and the completion-status of the status request. The completion routine must do a normal return when it completes.

completion-param

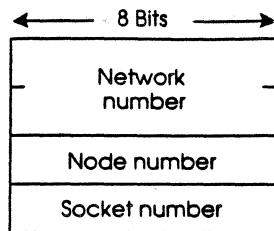
```
type:      longword
access:    read only
mechanism: by value
```

A caller-supplied value that is passed as an argument to the completion routine. The completion-param argument is a longword containing the caller-supplied value.

Description

A PAP client in a workstation calls this routine to determine the current status of a specified PAP server. The routine is executed asynchronously, which means control is returned to the caller as soon as the status request has been sent to the server. The library informs the client when the status is returned by the server through a client-supplied completion routine.

ATK\$PAP_STATUS requires the caller to supply the entity name of the server or the internet address of the server's service listening socket. If an entity name is supplied, **ATK\$NBP_LOOKUP** is used to find the internet address of the specified PAP server. The NBP call is issued with a transmission count and interval of 5. The internet address block is filled in with the PAP server's internet address so subsequent calls will not require the call to **ATK\$NBP_LOOKUP**. The internet address is supplied as an internet address block with the following format:



For more information, consult *Inside AppleTalk*.

Conditions	SS\$_NORMAL	Routine successfully completed.
	ATK__INVSTSRSP	Status request was responded to with an invalid PAP type code.

Any condition values returned by the following AppleTalk Protocol Library routines:

ATK\$NBP_LOOKUP

ATK\$ATP_SND_REQ

Any condition values returned by the following VMS Run-Time Library routines:

LIB\$GET_VM

ATK\$PAP_WRITE – Write over PAP connection

A PAP client at either end of a connection calls this routine to write data to the other end over the connection specified by the reference number.

Format

ATK\$PAP_WRITE **refnum**
 ,data-buf
 ,data-buf-size
 ,eof-flag
 ,completion-routine
 ,completion-param

Returns

type: longword (unsigned)
access: write only
mechanism: by value

One of the values listed below under "Conditions."

Arguments

refnum

type: longword (unsigned)
access: read only
mechanism: by value

The connection reference number of an opened PAP connection over which the data is to be sent. The **refnum** argument contains the value of the connection reference number.

data-buf

type: unspecified
access: read only
mechanism: by reference

The data to be sent to the other end of the PAP connection. The **data-buf** argument is the address of a buffer containing the data.

data-buf-size

type: word (unsigned)
access: read only
mechanism: by value

The size, in bytes (up to 512 * other end's flowquantum), of the data. The **data-buf-size** argument contains the value of the data size.

eof-flag

type: byte
access: read only
mechanism: by value

A flag value indicating whether or not the data is to be sent with the PAP end-of-file flag set. The eof-flag argument contains the value of the flag. Any nonzero value indicates end-of-file.

completion-routine

type: procedure entry mask
access: call without stack unwinding
mechanism: by reference

The entry mask of a procedure that is called when the send data request completes. The completion-routine argument contains the address of this procedure entry mask.

This routine is called with the following format:

Format	completion-routine	completion-parameter , completion-status
---------------	--------------------	---

Arguments	completion-parameter
------------------	----------------------

type:	longword
access:	read only
mechanism:	by value

completion-status

type:	longword
access:	read only
mechanism:	by value

The library calls the completion routine with the completion-parameter and the completion-status of the send data request. The completion routine must do a normal return when it completes.

completion-param

type: longword
access: read only
mechanism: by value

A caller-supplied value that is passed as an argument to the completion-routine argument. The completion-param argument is a longword containing the caller-supplied value.

Description

A PAP client at either end of a connection calls this routine to write data to the other end over the connection specified by the reference number. The routine is executed asynchronously, which means control is returned to the caller as soon as the request has been queued for sending. The library informs the caller of the request's completion through a client-supplied completion routine.

For more information, consult *Inside AppleTalk*.

Conditions	SS\$_NORMAL	Routine successfully completed.
	ATK__INVREFNUM	Invalid connection reference number.
	ATK__EXCFLWQTM	Exceeded the flowquantum of the other end of the connection.
	ATK__CONCLS	Request aborted by caller calling ATK\$PAP_CLOSE.
	ATK__CLSCONREQ	Request aborted by the other end requesting that the connection be closed.
	ATK__CONTMREXP	Request aborted as a result of the connection timer expiring.

Any condition values returned by the following AppleTalk Protocol Library routines:

ATK\$ATP_SND_RSP

Any condition values returned by the following VMS Run-Time Library routines:

LIB\$GET_VM

This routine removes an entry from a queue. **ATK\$REMQUE** makes the **VAX/VMS REMQUE** instruction available as a callable procedure.

Returns	type:	longword (unsigned)
	access:	write only
	mechanism:	by value

One of the values listed below under "Conditions."

Entry to be removed by `ATK$REMQUE`. The `entry` argument contains the address of this signed quadword-aligned queue entry.

```

type:      longword (unsigned)
access:    write only
mechanism: by reference

```

Address of removed entry. The `destination` argument is the address of an unsigned longword that receives the removed entry's address.

Conditions	SS\$ _NORMAL	Routine successfully completed.
	ATK _ONEENTQUE	Successful completion. The entry was removed from the queue, and the resulting queue is empty.
	ATK _QUEWASEMP	Unsuccessful completion. The queue was empty.

ATK\$RETURN_DATAGRAM – Return datagram to silo

This routine returns a datagram buffer to the silo list, making it available to receive incoming datagrams or for transmit allocation.

Format ATK\$RETURN_DATAGRAM datagram

Arguments **datagram**

type: unspecified
access: read only
mechanism: by reference

The datagram buffer to be returned to the silo list. The datagram argument contains the address of the datagram buffer.

Description This routine returns a datagram buffer to the "front" of the silo list.

ATK\$RTMP_REQ – Broadcast an RTMP request

This routine broadcasts an RTMP request packet to the RTMP listening socket in an attempt to discover a port's network number and the node ID of a bridge on the network.

Format `ATK$RTMP_REQ port-id
 [,thisnet
 [,trans-cnt
 [,trans-interval]]]`

Returns `type:            longword (unsigned)`
 `access:        write only`
 `mechanism:    by value`
One of the values listed below under "Conditions."

Arguments `port-id`
`type:            longword (unsigned)`
`access:        read only`
`mechanism:    by value`
The identifier of the port on which the datagram is to be transmitted. The `port-id` argument is the value of the port identifier.

thisnet

`type:            word`
`access:        write only`
`mechanism:    by reference`

The network number of the port. The `thisnet` argument is the address of a word to receive the network number of the port. If `thisnet` is not specified, the network number is still filled into the AppleTalk for VMS Protocol Library's internal port descriptor. Until a port's network number is known, the internet is effectively not accessible through the port.

trans-cnt

`type:            word`
`access:        read only`
`mechanism:    by value`

The number of times to broadcast the RTMP request packet. The `trans-cnt` argument contains the value of the transmission count. If `trans-cnt` is not specified or is zero, a default value of 4 is used.

trans-interval

type: word
access: read only
mechanism: by value

The interval, in seconds, between broadcast RTMP requests. The `trans-interval` argument contains the value of the transmission interval. If `trans-interval` is not specified or is zero, a default value of 1 second is used.

Description

This routine broadcasts an RTMP request packet addressed to the RTMP listening socket. A bridge process that receives an RTMP request will send back an RTMP response packet to the requestor's RTMP listening socket. The response packet provides the requestor with its network number and the node ID of the bridge process on the network. Until a port's network number is known, the internet is effectively not accessible through the port.

This routine is called for you by `ATK$OPEN_PORT` when an AppleTalk-speaking port is opened.

Conditions

Any condition values returned by the following AppleTalk Protocol Library routines:

`ATK$GET_DATAGRAM`

`ATK$DDP_WRITE`

`ATK$TMRINS`

ATK\$TMRINS – Install a timer process

This routine installs a timer process into the library's internal timer queue.

Format	ATK\$TMRINS	timer-id ,process ,process-parameter ,schedule-interval ,leave-on-queue
---------------	--------------------	--

Returns	type:	longword (unsigned)
	access:	write only
	mechanism:	by value

One of the values listed below under "Conditions."

Arguments	timer-id
	type: longword (unsigned)
	access: write only
	mechanism: by reference

A unique 32-bit identifier assigned to the timer process. The `timer-id` argument contains the address of the longword to receive the assigned timer ID.

process

```

type:      procedure entry mask
access:    call without stack unwinding
mechanism: by reference

```

The entry mask of a procedure that is called when timer schedule interval expires. The `process` argument contains the address of this procedure entry mask. The library calls the process with the `process-parameter` passed (by value) as an argument. The process must do a normal return when it completes.

process-parameter

```

type:      longword
access:    read only
mechanism: by value

```

A caller-supplied value that is passed as an argument to the timer process. The `process-parameter` argument is a longword containing the caller-supplied value.

schedule-interval

```

type:      word
access:    read only
mechanism: by value

```

The delta time, in seconds, when the timer process is to be scheduled for execution. The `schedule-interval` argument is the value of the scheduling time.

leave-on-queue

type: byte
access: read only
mechanism: by value

A flag value indicating whether or not the timer process is to be automatically rescheduled after execution by being left on the library's timer queue. The `leave-on-queue` argument contains the value of the flag. Any nonzero value indicates that the timer process is to be left on the timer queue. A timer process inserted with a nonzero `leave-on-queue` flag can only be removed by calling `ATK$TMRREM`.

Description

This routine installs a timer process on the library's internal timer queue. This causes the timer process to be scheduled for execution when the specified schedule interval expires.

The call to `ATK$INIT_ATALK` initializes the library's internal timer process scheduler. The VMS `SYS$SETIMR` system service is used to schedule the execution of the library's process scheduler. This process is scheduled for execution every 1/2 second. This means that the first time an installed timer process is scheduled for execution, it may actually be up to a half second prior to its actual specified schedule interval.

The library uses a timer ID of 1 for the `SYS$SETIMR` call. This timer must not be canceled by calling the VMS `SYS$CANTIM` system service. Note that calling `SYS$CANTIM` without specifying a timer ID or specifying a zero for the timer ID causes all `SYS$SETIMR` timers to be canceled.

Timer processes should be designed to use a minimum amount of CPU time. They are called at AST level, and therefore any completion routines or asynchronous activity are disabled while a timer process is being executed. In addition, the library uses the same timer queue for its internal timer processes.

Conditions

`SS$_NORMAL` Routine successfully completed.

Any condition values returned by the following VMS Run-Time Library routines:

`LIB$GET_VM`

ATK\$TMRREM – Remove a timer process

This routine removes a timer process from the library's internal timer queue.

Format **ATK\$TMRREM** **timer-id**

Returns type: longword (unsigned)
 access: write only
 mechanism: by value

One of the values listed below under "Conditions."

Arguments **timer-id**

type: longword (unsigned)
access: read only
mechanism: by value

A unique 32-bit identifier assigned to the timer process. The `timer-id` argument contains the value of the `ATK$TMRINS` assigned longword timer ID.

Conditions **SS\$_NORMAL** Routine successfully completed.

ATK\$TMRRESET – Reset a timer process

This routine resets an installed timer process to its initial schedule interval.

Format ATK\$TMRRESET timer-id

Returns type: longword (unsigned)
 access: write only
 mechanism: by value

One of the values listed below under “Conditions.”

Arguments timer-id

 type: longword (unsigned)
 access: read only
 mechanism: by value

A unique 32-bit identifier assigned to the timer process. The timer-id argument contains the value of the ATK\$TMRINS assigned longword timer ID.

Conditions SS\$_NORMAL Routine successfully completed.

ATK\$VA_COUNT – Variable argument count

This routine returns the number of arguments specified in the argument list to the caller's caller; in other words, the number of arguments "you" were called with.

Format

ATK\$VA_COUNT

Returns

type: longword (unsigned)
access: write only
mechanism: by value

Number of arguments in the caller's caller argument list.

ATK\$ZIP_BRINGUP – Bring up a network

This routine requests that a network be brought back up. It instructs all bridges receiving the request to insert the network number into their routing tables and assign it the specified zone name.

Format `ATK$ZIP_BRINGUP    zone-dsc`

Returns type: longword (unsigned)
 access: write only
 mechanism: by value

One of the values listed below under "Conditions."

Arguments `zone-dsc`

type: character string
access: read only
mechanism: by descriptor

The zone name to assign the network that is being brought back up. The `zone-dsc` argument is the address of a character string descriptor containing the length of the zone name and a pointer to it.

Description This routine broadcasts a ZIP bringup request over an end node's directly connected network. Any bridge receiving the request adds the network number to its routing table and assigns it the specified zone name.

Note that this request may be issued only by end nodes on the target network.

Conditions `SS$_NORMAL` Routine successfully completed.

Any condition values returned by the following AppleTalk Protocol Library routines:

`ATK$GET_DATAGRAM`

`ATK$DDP_OPN_SKT`

`ATK$DDP_WRITE`

Any condition values returned by the following VMS Run-Time Library routines:

`LIB$SCOPY_DXDX`

This routine obtains the zone name for a given port.

Returns type: longword (unsigned)
 access: write only
 mechanism: by value

One of the values listed below under "Conditions."

```

zone
type:      unspecified
access:    write only
mechanism: by reference

```

The zone name of the specified port. The zone argument contains the address of an array of bytes large enough for a count byte plus the maximum zone name length of 32 bytes.

Conditions	SS\$ _NORMAL	Routine successfully completed.
	ATK NOBRIDGE	No bridge is available to service the request.

Any condition values returned by the following AppleTalk Protocol Library routines:

ATK\$ATP SND REQ

ATK\$ZIP_GET_ZONE_LIST – Get list of zone names in internet

This routine obtains a complete list of zone names defined in the internet.

Format

```
ATK$ZIP_GET_ZONE_LIST  port-id
                        ,start-index
                        ,zone-list
                        ,num-of-zones
                        ,last-flag
```

Returns

type: longword (unsigned)
access: write only
mechanism: by value

One of the values listed below under "Conditions."

Arguments

port-id

type: longword (unsigned)
access: read only
mechanism: by value

The identifier of the port on which to issue the get zone list request. The `port-id` argument contains the value of the port identifier.

start-index

type: word (unsigned)
access: read only
mechanism: by value

The starting index for the get zone list request. The `start-index` is the value of the index at which to start including zone names in the response. It is used to obtain a zone list that may not have fit in one ATP response packet. The first call to get zone list should specify a `start-index` of 1. While `last-flag` is not returned as minus 1, the caller must reissue the get zone list request and specify a `start-index` equal to the previous returned value of `num-of-zones` plus 1.

zone-list

type: unspecified
access: write only
mechanism: by reference

The list of zone names in the internet. The `zone-list` argument contains the address of a buffer that is filled with the list of zone names. Each zone name is preceded by a single byte-length field. The zone list buffer must be at least the size of the largest ATP request packet (578 bytes).

num-of-zones

type: word (unsigned)
access: write only
mechanism: by reference

The number of zone names in the list. The `num-of-zones` argument contains the address of a word that receives the number of zone names in the zone list buffer.

last-flag

type: byte (unsigned)
access: write only
mechanism: by reference

A flag indicating whether or not this is the last zone in the bridge's Zone Information Table (ZIT). The `last-flag` argument contains the address of a byte that is set to minus 1 if the `zone-list` argument has the last zone name from the bridge's ZIT.

Description

This routine sends a ZIP request to a bridge for the list of zone names in the internet. The list is filled into the supplied buffer as concatenated counted strings. See the appropriate AppleTalk ZIP documentation for usage information.

Conditions

`SS$_NORMAL` Routine successfully completed.

`ATK__NOBRIDGE` No bridge is available to service the request.

Any condition values returned by the following AppleTalk Protocol Library routines:

`ATK$ATP_SND_REQ`

ATK\$ZIP_QUERY – Send network zone name query

This routine sends a ZIP query message requesting the zone names of the specified list of networks.

```
Format      ATK$ZIP_QUERY      port-id
              ,nets
              ,net-cnt
```

Returns	type:	longword (unsigned)
	access:	write only
	mechanism:	by value

One of the values listed below under "Conditions."

Arguments	port-id
	type: longword (unsigned)
	access: read only
	mechanism: by value

The identifier of the port on which to issue the zone query. The `port-id` argument contains the value of the port identifier.

```

nets
type:      word (unsigned)
access:    read only
mechanism: by reference

```

The list of network numbers whose zone name is to be queried. The `net:s` argument contains the address of the list of network numbers. Each network number is a 2-byte number.

```
net-cnt
type:      byte (unsigned)
access:    read only
mechanism: by value
```

The number of networks in the nets list. The `net_cnt` argument contains the value of the number of networks in the list.

Description	This routine sends a ZIP query for the zone names of a list of networks. The caller is required to have established the zone information socket listener that will receive the replies.
--------------------	---

- ❖ *Note:* This routine should be used only by bridge and net management processes. User processes should use the `ATK$GET_ZONE_LIST` and `ATK$GET MY_ZONE` procedures.

Conditions

SS\$ _NORMAL

Routine successfully completed.

Any condition values returned by the following AppleTalk Protocol Library routines:

ATK\$GET_DATAGRAM

ATK\$LAP_XMIT

ATK\$ZIP_TAKEDOWN - Take down a network

This routine requests that a network be taken down for the purpose of changing its zone name.

Format ATK\$ZIP_TAKEDOWN

Returns type: longword (unsigned)
 access: write only
 mechanism: by value

One of the values listed below under "Conditions."

Description This routine broadcasts a ZIP takedown request over an end node's directly connected network.

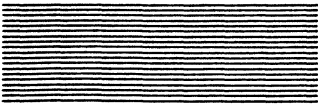
Conditions SS\$_NORMAL Routine successfully completed.

Any condition values returned by the following AppleTalk Protocol Library routines:

ATK\$GET_DATAGRAM

ATK\$DDP_OPN_SKT





Appendix A



Port Drivers

This appendix contains descriptions of the various port drivers currently supported by the AppleTalk for VMS Protocol Library.

The port drivers are implemented as separate, loadable, shareable images. The port driver images are loaded by calling the `ATK$INIT_PORT` routine with the name of the port driver. The `ATK$OPEN_PORT` routine activates the port driver.

The `ATK$INIT_PORT` routine uses the VAX/VMS Run-Time Library routine `LIB$FIND_IMAGE_SYMBOL` to dynamically activate the port driver shareable images. The `ATK$OPEN_PORT` routine returns a port ID that must be used by the client in all future references to the port.

Each port driver may require its own port-specific information. This information is supplied to the `ATK$INIT_PORT` and `ATK$OPEN_PORT` routines in the port information string descriptor parameter. When the port driver is loaded, the port information string is passed to the port driver.

ATKAHLAP – Asynchronous half-bridge port driver

The asynchronous half-bridge port driver is responsible for AppleTalk communication between the AppleTalk for VMS Protocol Library and a port connected to a VMS terminal device. This type of port is useful only to bridge processes for communication with other bridges using the half-bridge protocol.

Format `ATK$OPEN_PORT port-id
 ,portdriver
 ,portinfo
 ,portname`

Returns type: longword (unsigned)
 access: write only
 mechanism: by value

One of the values listed below under "Conditions."

Arguments **port-id**
 type: longword (unsigned)
 access: write only
 mechanism: by reference

The numeric identifier to be used with this port. The `port-id` argument is the address of an unsigned longword that receives the port ID.

portdriver
type: character string
access: read only
mechanism: by descriptor

The AHLAP port driver shareable image file name subject to the restrictions and requirements of the VMS Run-Time Library routine `LIB$FIND_IMAGE_SYMBOL`. The `portdriver` argument is the address of a character string descriptor pointing to this string. The AHLAP port driver file name is `ATKAHLAP.EXE`. This argument may contain only the file name. File type cannot be specified.

portinfo
type: character string
access: read only
mechanism: by descriptor

The AHLAP port information string to be passed to the AHLAP port driver open routine. The `portinfo` argument is the address of a character string descriptor pointing to this string. For the AHLAP port driver, this string contains the name of the asynchronous terminal device to be used for half-bridge communication and the baud rate to which the specified asynchronous device will be set, if possible, before starting communication. The device name and baud rate must be separated by a comma. Logical translations are performed on the supplied device name.

Permissible baud rate values are 0, 1200, 2400, 4800, 9600, and 19200. If zero is specified for the baud rate, and the asynchronous device is not set to "autobaud," the permanent device speed setting is used. If the device is set to autobaud, the speed is set to 2400.

portname

type: character string
access: read only
mechanism: by descriptor

The name to be assigned to the port. The **portname** argument is the address of a descriptor pointing to this string.

Description

The asynchronous half-bridge port driver is responsible for AppleTalk communication between the AppleTalk for VMS Protocol Library and a port connected to a VMS terminal device. This type of port is useful only to bridge processes for communication with other bridges. The link-level protocol is compatible with the Hayes InterBridge but is not officially recognized by Apple Computer.

Conditions

SS\$_NORMAL Routine successfully completed.

ATK\$_ONEENTQUE Routine completed successfully; however, only one entry remaining.

ATK\$_QUEWASEMP There are no buffers available for allocation.

Any condition values returned by the following VMS system services:

SYS\$ASSIGN

SYS\$QIOW

Any I/O error codes returned by the various asynchronous device drivers.

Any condition values returned by the following AppleTalk Protocol Library routines:

ATK\$ LAP_XMIT

Any condition values returned by the following VMS Run-Time Library routines:

LIB\$GET_VM

LIB\$SCOPY_DXDX

ATKELAP – Ethernet port driver

The Ethernet port driver is responsible for all communication between the AppleTalk for VMS Protocol Library and the VMS Ethernet XE: or XQ: drivers.

Format

```
ATK$OPEN_PORT    port-id
                  ,portdriver
                  ,portinfo
                  ,portname
```

Returns

type: longword (unsigned)
access: write only
mechanism: by value

One of the values listed below under "Conditions."

Arguments

port-id

type: longword (unsigned)
access: write only
mechanism: by reference

The numeric identifier to be used with this port. The `port-id` argument is the address of an unsigned longword that receives the port ID.

portdriver

type: character string
access: read only
mechanism: by descriptor

The ELAP port driver shareable image file name subject to the restrictions and requirements of the VMS Run-Time Library routine `LIB$FIND_IMAGE_SYMBOL`. The `portdriver` argument is the address of a character string descriptor pointing to this string. The ELAP port driver file name is `ATKELAP.EXE`. This argument may contain only the file name. File type cannot be specified.

portinfo

type: character string
access: read only
mechanism: by descriptor

The ELAP port information string to be passed to the ELAP port driver open routine. The `portinfo` argument is the address of a descriptor pointing to this string. For the ELAP port driver, this string is the single ASCII character representation of the controller index ('A' through 'Z') of the particular DEUNA/DELUA/DEQNA controller to use. If this argument is zero, the ELAP port driver will use controller 'A'.

portname

type: character string
access: read only
mechanism: by descriptor

The name to be assigned to the port. The portname argument is the address of a descriptor pointing to this string.

Description

The ELAP port driver establishes AppleTalk communication between the calling process and a port connected to an Ethernet backbone network. This port type is useful only to bridge processes and other communication-oriented applications, since the Ethernet is not DDP-addressable.

The ELAP port driver attempts to initialize both XE: and XQ: devices. Therefore, an application written to run on Unibus systems will run on Q-bus systems without modification.

The XE: or XQ: device is initialized with eight preallocated 1500-byte receive buffers. The receive buffers are subject to the process's buffered I/O byte count quota. If the allocation of the receive buffers fails because of lack of buffered I/O byte count quota, the XE: or XQ: device drivers return a status of \$\$\$_BADPARAM.

Conditions

\$\$\$_NORMAL Routine successfully completed.

ATK\$_ONEENTQUE Routine completed successfully; however, only one entry remaining.

ATK\$_QUEWASEMP There are no buffers available for allocation.

Any condition values returned by the following VMS system services:

SYSS\$ASSIGN

SYSS\$QIOW

Any I/O error codes returned by the XE: or XQ: device drivers.

Any condition values returned by the following AppleTalk Protocol Library routines:

ATK\$SLAP_XMIT

Any condition values returned by the following VMS Run-Time Library routines:

LIB\$GET_VM

ATKVLAP – Virtual AppleTalk port driver

The virtual AppleTalk port driver is responsible for all communication between the AppleTalk for VMS Protocol Library and the VMS virtual AppleTalk network provided by the VN: device driver.

Format

```
ATK$OPEN_PORT    portid
                  ,portdriver
                  ,portinfo
                  ,portname
```

Returns

type: longword (unsigned)
access: write only
mechanism: by value

One of the values listed below under "Conditions."

Arguments

port-id

type: longword (unsigned)
access: write only
mechanism: by reference

The numeric identifier to be used with this port. The `port-id` argument is the address of an unsigned longword that receives the port ID.

portdriver

type: character string
access: read only
mechanism: by descriptor

The VLAP port driver shareable image file name subject to the restrictions and requirements of the VMS Run-Time Library routine `LIB$FIND_IMAGE_SYMBOL`. The `portdriver` argument is the address of a character string descriptor pointing to this string. The VLAP port driver file name is `ATKVLAP.EXE`. This argument may contain only the file name. File type cannot be specified.

portinfo

type: character string
access: read only
mechanism: by descriptor

The VLAP port information string to be passed to the VLAP port driver open routine. The `portinfo` argument is the address of a character string descriptor pointing to this string. For the VLAP port driver, this string contains the AppleTalk network number (in hexadecimal) to assign to the VLAP port, and the controller ID, in the following format:

`networknumber,controllerID`

Normally, the AppleTalk network number is only assigned by bridge processes, and this argument should be zero when opening a VLAP port from an end-node process. If the port information argument is zero, the VLAP port driver will not assign a network number to the port. The controller ID is optional, and will default to controller A if missing.

portname

type: character string
 access: read only
 mechanism: by descriptor

The name to be assigned to the port. The portname argument is the address of a descriptor pointing to this string.

Conditions

SS\$_NORMAL Routine successfully completed.

ATK\$_ONEENTQUE Routine completed successfully; however, only one entry remaining.

ATK\$_QUEWASEMP There are no buffers available for allocation.

Any condition values returned by the following VMS system services:

SYSS\$ASSIGN

SYSS\$QIOW

SYSS\$GETDVI

Any I/O error codes returned by the VN: device driver.

Any condition values returned by the following VMS Run-Time Library routines:

LIB\$GET_VM

LIB\$SCOPY_DXDX

SYS\$QIO
LIB\$GET VM

The following format is used to open the DLAP port driver:

Format	ATK\$OPEN_PORT	portid , portdriver , portinfo , portname
---------------	----------------	--

Returns	type:	longword (unsigned)
	access:	write only
	mechanism:	by value

One of the values listed below under "Conditions."

Arguments	port-id
type:	longword (unsigned)
access:	write only
mechanism:	by reference

The numeric identifier to be used with this port. The `port-id` argument is the address of an unsigned longword that receives the port ID.

portdriver

```
type:      character string
access:    read only
mechanism: by descriptor
```

The DLAP port driver shareable image file name subject to the restrictions and requirements of the VMS Run-Time Library routine `LIB$FIND_IMAGE_SYMBOL`. The `portdriver` argument is the address of a character string descriptor pointing to this string. The DLAP port driver file name is `ATKDLAP.EXE`. The argument may contain only the file name. File type cannot be specified.

portinfo

```

type:      character string
access:    read only
mechanism: by descriptor

```

The DLAP port information string to be passed to the DLAP port driver initialization routine. The `portinfo` argument is the address of a character string descriptor pointing to this string. For the DLAP port driver, this string has the following format:

nodename, objectname

where `nodename` is the DECnet node name to which the port driver will establish communication. The `objectname` is the DECnet object name registered for the port driver on the specified node (see `ATK$INIT_PORT`, above). If `objectname` is blank, a default object name "ATKBRIDGE" is used.

The node name must be specified, but the object name is optional. If the object name is not supplied, there should be no comma in the string.

portname

type: character string
access: read only
mechanism: by descriptor

The user-supplied name for the opened DLAP port. The portname argument is the address of a character string descriptor pointing to this string.

Conditions

SS\$ _NORMAL Routine successfully completed.

ATK__PRTNOTINIT Port has not been initialized with ATK\$INIT_PORT.

Any condition values returned by the following VMS Run-Time Library routines:

LIB\$COPY_DXDX

LIB\$ANALYZE_DESC

LIB\$GET_VM

LIB\$FREE_VM

SYSS\$ASSIGN

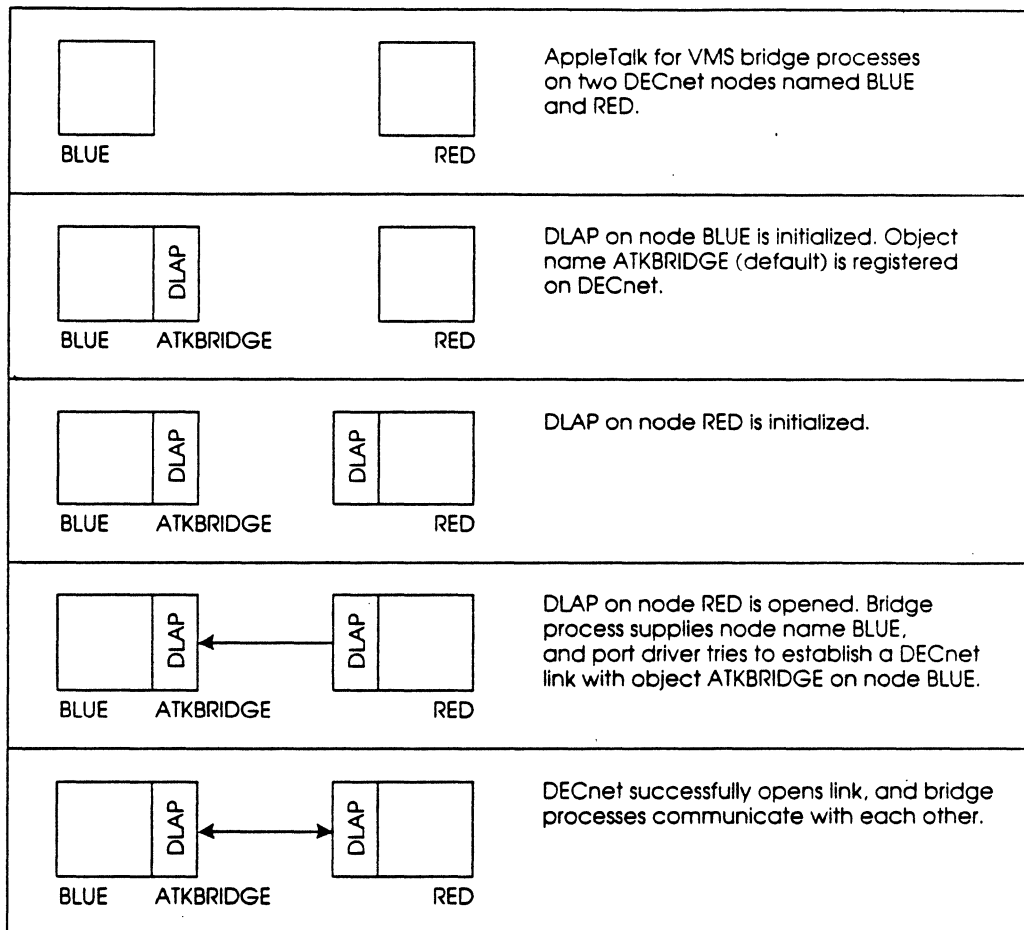
SYSS\$GETDVI

SYSS\$QIO

Description

The DLAP port driver manages AppleTalk communication between the AppleTalk for VMS Protocol Library and a port connected to a DECnet link. This type of port is useful only to bridge processes for communication with other bridge processes using DECnet as the link between them.

It is important to note the sequence for initializing and opening cooperating DLAP ports on DECnet nodes. A DLAP port driver must be initialized, but not opened, on a node before processes using DLAP port drivers on other DECnet nodes can communicate with it. This is shown in the following diagram.



When the DLAP port driver is initialized, it will register itself as a DECnet object using the name supplied in the `port info` parameter. When a DLAP port driver is initialized and opened on another DECnet node, this object name is used to establish communication through DECnet (see the `ATK$OPEN_PORT` call, above).

The default object name is `ATKBRIDGE`. This default name is used if no object name is supplied by the `port info` parameter.



Index

A

address (AppleTalk), get 77
AHLAP (asynchronous half-bridge port driver) 16, 17, 124-125
ALAP 15
AppleTalk, initialize 57-58
AppleTalk address, get 77
AppleTalk port driver. *See* VLAP
AppleTalk Transaction Protocol. *See* ATP
argument passing 8-9
argument count (variable) 114
asynchronous half-bridge port driver. *See* AHLAP
ATKAHLAP 124-125
ATK\$ATP_CANCEL_REQ 21
ATK\$ATP_CANCEL_RSP 22
ATK\$ATP_CLS_SKT 23
ATK\$ATP_GET_REQ 24-27
ATK\$ATP_OPN_SKT 28
ATK\$ATP_SND_REQ 29-33
ATK\$ATP_SND_RSP 34-38
ATK\$BYTESWAP 39-40
ATK\$CHECKSUM 41
ATK\$CLOSE_PORT 42
ATK\$DDP_CLS_SKT 43
ATK\$DDP_OPN_SKT 44-45
ATK\$DDP_READ 46
ATK\$DDP_WRITE 47-50
ATKDLAP 124-125
ATKELAP 126-127
ATK\$ENTITY_LEN 51
ATK\$SEP_REQ 52-54
ATK\$GET_DATAGRAM 55
ATK\$GET_PHYS_ADDR 56
ATK\$INIT_ATALK 12-13, 57-58
ATK\$INIT_PORT 59
ATK\$INSQUE 60
ATK\$LAP_XMIT 61-63
ATK\$MAKE_ENTITY 64-65
ATK\$NBP_CONFIRM 66-68
ATK\$NBP_LOOKUP 69-72
ATK\$NBP_REGISTER 73-74
ATK\$NBP_REMOVE 75-76
ATK\$NET_NODE 77

ATK\$OPEN_PORT 78-79
ATK\$PAP_CLOSE 80
ATK\$PAP_OPEN 81-86
ATK\$PAP_READ 87-89
ATK\$PAP_REG_NAME 90-91
ATK\$PAP_REM_NAME 92
ATK\$PAP_SLINIT 93-98
ATK\$PAP_STATUS 99-102
ATK\$PAP_WRITE 103-105
ATK\$REMQUE 106
ATK\$RETURN_DATAGRAM 107
ATK\$RTMP_REQ 108-109
ATK\$TMRINS 110-111
ATK\$TMRREM 112
ATK\$TMRRESET 113
ATK\$VA_COUNT 114
ATKVLAP 127-128
ATK\$ZIP_BRINGUP 115
ATK\$ZIP_GET_MY_ZONE 116
ATK\$ZIP_GET_ZONE_LIST 117-118
ATK\$ZIP_QUERY 119-120
ATK\$ZIP_TAKEDOWN 121
ATP (AppleTalk Transaction Protocol) 5
ATP request
cancel 21
receive 24-27
send 29-33
ATP response
cancel 22
send 34-38
ATP socket
close 23
open 28

B

bridges 14
bring up network 115
broadcast RTMP request 108-109
buffer (datagram), get 55
buffer pool 12-14
byte swapper 39-40

C

call formats 7
calling Protocol Library
procedures 6-10
cancel
ATP request 21
ATP response 22
checksum (DDP), compute 41
C language 7
close
ATP socket 23
DDP socket 43
PAP connection 80
port driver 42
communicating on ports 16-17
completion routine 14
compute DDP checksum 41
condition codes 8
confirm name via NBP 66-68
connection (PAP)
close 80
open 81-86
read over 87-89
write over 103-105
construct NBP entity 64-65

D

datagram buffers 12-14
get 55
Datagram Delivery Protocol. *See* DDP
datagrams
incoming 13-14
read 46
return to silo 107
send 14
transmit 47-50, 61-63
DDP (Datagram Delivery Protocol) 5
DDP checksum, compute 41
DDP datagrams. *See* datagrams
DDP socket
close 43
open 44-45

DECnet half-bridge port driver. *See* DLAP
descriptor, passing arguments by 8,
9

DEUNA/DEQNA driver
(Ethernet) 12

device types 15-16

DLAP (DECnet half-bridge port
driver) 16, 17, 130-133

drivers. *See* port drivers

dynamic memory allocation 10

E, F, G, H

Echo Protocol. *See* EP

ELAP (Ethernet port driver) 16,
17, 126-127

entry

insert in queue 60

remove from queue 106

EP (Echo Protocol) 5

EP request, send 52-54

error conditions 10

Ethernet DEUNA/DEQNA
driver 12

Ethernet port driver. *See* ELAP
event flags 9

I, J, K

incoming datagrams 13

initialize

AppleTalk 57-58

PAP server 93-98

port driver 59

insert entry in queue 60

install timer process 110-111

International Standards Organization
(ISO), Open Systems

Interconnection (OSI) model 5

internet, get zone name list
in 117-118

L

LIB\$FIND_IMAGE_SYMBOL 15

LIB\$GET_EF 19

LIB\$GET_VM 10

linking with Protocol Library 10

link-level communication, buffer
pool and 12-14

load port driver 59-60

look up name via NBP 69-72

low-level I/O facilities 12-17

M

memory allocation, dynamic 10

Modular Programming Standard
(VAX) 4

N

name

confirm via NBP 66-68

look up via NBP 69-72

register via NBP 73-74

remove NBP 75-76

Name Binding Protocol. *See* NBP
naming conventions 7-8

NBP (Name Binding Protocol) 5

confirm name via 66-68

look up name via 69-72

register name via 73-74

NBP entity

construct 64-65

return length of 51

NBP name, remove 75-76

network

bring up 115

take down 121

network zone name query,
send 119-120

O

open

ATP socket 28

DDP socket 44-45

PAP connection 81-86

port 78-79

Open Systems Interconnection

(OSI) model, International
Standards Organization (ISO) 5

P

PAP (Printer Access Protocol) 5

PAP connection

close 80

open 81-86

read over 87-89

write over 103-105

PAP server

initialize 93-98

request status 99-102

passing arguments 8-9

physical address (port), get 56

port drivers 15, 123-133

AHLAP 124-125

close 42

DLAP 130-133

ELAP 126-127

initialize 59

load 59

VLAP 128-129

port physical address, get 56

ports 14

communicating on 16-17

open 78-79

port zone name, get 116

Printer Access Protocol. *See* PAP
procedure, defined 4

Protocol Library

calling procedures 6-10

error conditions and 10

features of 4-5

linking with 10

organization of 5-6

overview 4-17

routines 4, 19-121

using 4

Q

queue

insert entry in 60

remove entry from 106

R

read

datagrams 46

over PAP connection 87-89

receive ATP request 24-27

reference, passing arguments by 8,
9

register

name via NBP 73-74

server name 90-91

remove

entry from queue 106

NBP name 75-76

server name 92

timer process 112

request PAP server status 99-102

reset timer process 113

return

datagram to silo 107

length of NBP entity 51

routines. *See* Protocol Library or
specific routine

RTMP (Routing Table Maintenance
Protocol) 5

RTMP request, broadcast 108-109

S

send

ATP request 29-33

ATP response 34-38

datagrams 14

EP request 52-54

network zone name

query 119-120

server (PAP)

- initialize 93–98
- request status 99–102
- server name
 - register 90–91
 - remove 92
- silo, return datagram to 107
- silo list 13–14
- socket (DDP)
 - close 43
 - open 44–45
- socket listener procedure 13–14
- status codes 8

T, U

- take down network 121
- timer process
 - install 110–111
 - remove 112
 - reset 113
- transmit datagrams 47–50, 61–63
- transmit requests 14

V

- value, passing arguments by 8, 9
- variable argument count 114
- VAX Modular Programming Standard 4

VLAP (virtual AppleTalk port driver) 14, 16, 129–129

W, X, Y

- write over PAP
 - connection 103–105

Z

- ZIP (Zone Information Protocol) 5
- zone name (port), get 116
- zone name list, get in
 - internet 117–118
- zone name query (network),
 - send 119–120

THE APPLE PUBLISHING SYSTEM

This Apple manual was written, edited, and composed on a desktop publishing system using the Apple Macintosh™ Plus and Microsoft® Word. Proof and final pages were created on the Apple LaserWriter® Plus. POSTSCRIPT®, the LaserWriter page-description language, was developed by Adobe Systems Incorporated.

Text type is ITC Garamond® (a downloadable font distributed by Adobe Systems). Display type is ITC Avant Garde Gothic®. Bullets are ITC Zapf Dingbats®. Some elements, such as program listings are set in Apple Courier, a fixed-width font.